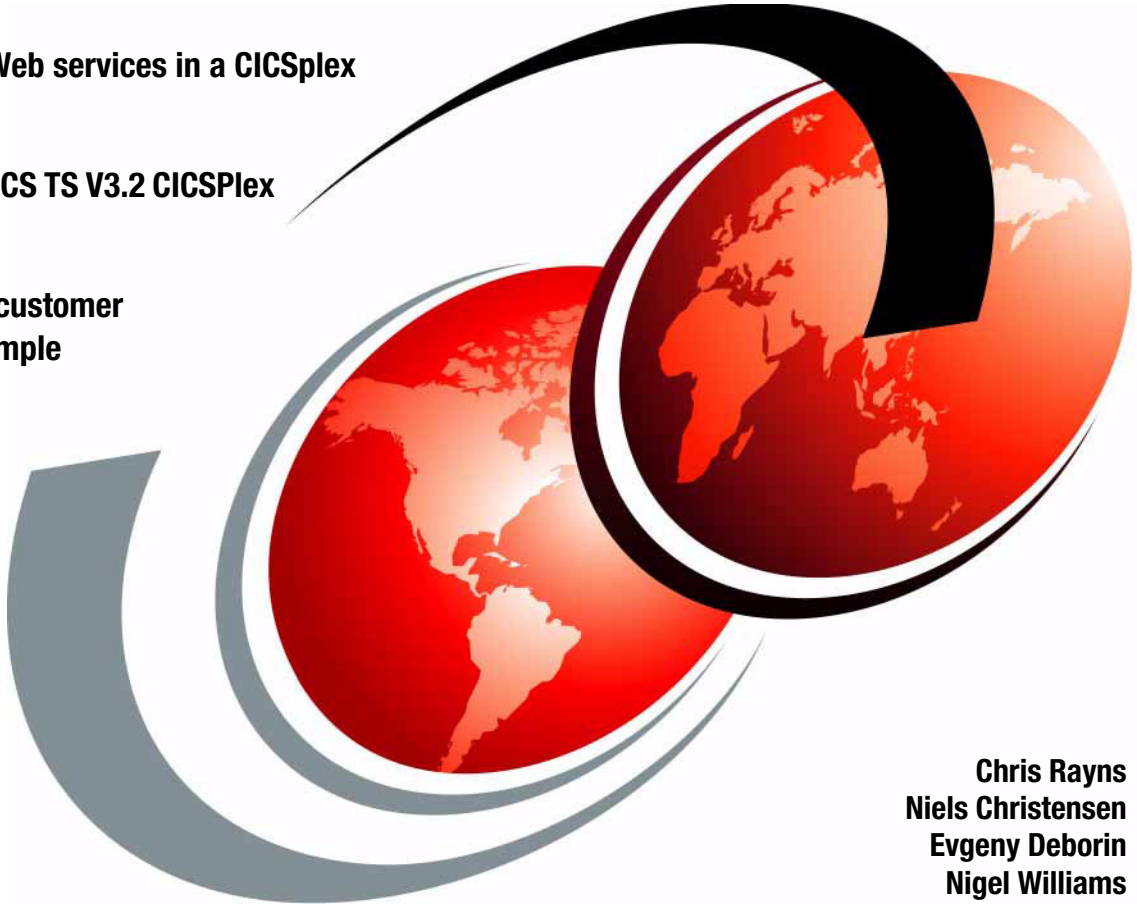
IBM

CICS Web Services Workload Management and Availability

- Deploying Web services in a CICSplex
- Based on CICS TS V3.2 CICSPlex SM
- Contains a customer project example



Chris Rayns
Niels Christensen
Evgeny Deborin
Nigel Williams

Redbooks

ibm.com/redbooks

International Technical Support Organization

CICS Web Services Workload Management and Availability

March 2008

SG24-7144-01

Note: Before using this information and the product it supports, read the information in “Notices” on page vii.

Second Edition (March 2008)

This edition applies to Version 3, Release 2, CICS Transaction Server

Contents

Notices

This information was developed for products and services offered in the U.S.A.

IBM may not offer the products, services, or features discussed in this document in other countries. Consult your local IBM representative for information on the products and services currently available in your area. Any reference to an IBM product, program, or service is not intended to state or imply that only that IBM product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any IBM intellectual property right may be used instead. However, it is the user's responsibility to evaluate and verify the operation of any non-IBM product, program, or service.

IBM may have patents or pending patent applications covering subject matter described in this document. The furnishing of this document does not give you any license to these patents. You can send license inquiries, in writing, to:
IBM Director of Licensing, IBM Corporation, North Castle Drive, Armonk, NY 10504-1785 U.S.A.

The following paragraph does not apply to the United Kingdom or any other country where such provisions are inconsistent with local law: INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PUBLICATION "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. Some states do not allow disclaimer of express or implied warranties in certain transactions, therefore, this statement may not apply to you.

This information could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be incorporated in new editions of the publication. IBM may make improvements and/or changes in the product(s) and/or the program(s) described in this publication at any time without notice.

Any references in this information to non-IBM Web sites are provided for convenience only and do not in any manner serve as an endorsement of those Web sites. The materials at those Web sites are not part of the materials for this IBM product and use of those Web sites is at your own risk.

IBM may use or distribute any of the information you supply in any way it believes appropriate without incurring any obligation to you.

Information concerning non-IBM products was obtained from the suppliers of those products, their published announcements or other publicly available sources. IBM has not tested those products and cannot confirm the accuracy of performance, compatibility or any other claims related to non-IBM products. Questions on the capabilities of non-IBM products should be addressed to the suppliers of those products.

This information contains examples of data and reports used in daily business operations. To illustrate them as completely as possible, the examples include the names of individuals, companies, brands, and products. All of these names are fictitious and any similarity to the names and addresses used by an actual business enterprise is entirely coincidental.

COPYRIGHT LICENSE:

This information contains sample application programs in source language, which illustrate programming techniques on various operating platforms. You may copy, modify, and distribute these sample programs in any form without payment to IBM, for the purposes of developing, using, marketing or distributing application programs conforming to the application programming interface for the operating platform for which the sample programs are written. These examples have not been thoroughly tested under all conditions. IBM, therefore, cannot guarantee or imply reliability, serviceability, or function of these programs.

Trademarks

The following terms are trademarks of the International Business Machines Corporation in the United States, other countries, or both:

AIX®
CICS®
CICSPlex®
DataPower®
DB2®
IBM®
IMS™
MVS™
OMEGAMON®
OS/390®
Parallel Sysplex®
Rational®
Redbooks®
Redbooks (logo)  ®
RACF®
REXX™
RMF™
S/390®
SecureWay®
SupportPac™
System z™
Tivoli®
WebSphere®
z/OS®
zSeries®

The following terms are trademarks of other companies:

Java, JVM, J2EE, and all Java-based trademarks are trademarks of Sun Microsystems, Inc. in the United States, other countries, or both.

Internet Explorer, Microsoft, Windows, and the Windows logo are trademarks of Microsoft Corporation in the United States, other countries, or both.

UNIX is a registered trademark of The Open Group in the United States and other countries.

Linux is a trademark of Linus Torvalds in the United States, other countries, or both.

Other company, product, or service names may be trademarks or service marks of others.

Preface

A key feature of CICS® TS V3 is its ability to use CICS to provide an execution environment for Web services applications.

This IBM® Redbooks® publication contains the specific management requirements of such applications and shows how CICSPlex® SM can be used to manage these applications in an easy to use fashion.

We first take a theoretical approach by looking at an overview of Web services, focusing on some of the architectural concepts that need to be considered in a Web services project. We define and discuss *service-oriented architecture* (SOA) and the relationship between SOAs and Web services.

We also discuss how CICS implements Web services by reviewing the support for SOAP that CICS provides in the SOAP for CICS feature with CICS TS V2. Then we discuss the Web services support of CICS TS V3.1 and the enhancements provided with CICS TS V3.2.

We then give a small overview of CICSPlex SM and how the CICSPlex SM WUI can assist with both CICS Web Services installation and management.

We next discuss the different techniques that can be used to provide high system availability and workload management for CICS Web service applications.

Finally, we discuss how high availability is provided across a Parallel Sysplex® where TCP/IP traffic uses the following functions:

- Port Sharing
- Virtual IP Addressing
- Sysplex Distributor

We then take a more hands on approach. We describe how CICSPlex SM is used to define and manage the required CICS definitions to run the Web services application and configure CICS as a service provider.

We then show how to set up a CICS Web services requester application for high availability. For this purpose, we use the CICS sample catalog application

Finally, we describe a practical example of a CICS Web services implementation based on a proof of concept for a large financial group, which was carried out in the IBM Customer Support Center at Montpelier, France.

The team that wrote this book

This book was produced by a team of specialists from around the world working at the International Technical Support Organization, Raleigh Center.

Chris Rayns is an IT Specialist and Project Leader at the ITSO, Poughkeepsie Center in New York. Chris writes extensively on all areas of S/390® security. Before joining the ITSO, he worked in IBM Global Services in the United Kingdom (UK) as a CICS IT Specialist.

Niels Christensen is an IT Specialist working as a Technical Team Lead for CICS and IMS™ at IBM IT Delivery Denmark. He has more than 20 years of experience with CICS, and specializes in availability and performance.

Evgeny Deborin is a group leader with the WebSphere® Technical Sales team working for IBM Israel. He is an IBM Certified Solution Designer for CICS Enablement for e-business and an IBM Certified System Administrator for WebSphere Application Server for z/OS®. Evgeny was twice on ITSO assignments (1997-99 and 2001-03) running a variety of projects in the area of transaction management systems (CICS Transaction Server, WebSphere Application Server, and WebSphere MQ). He ran ITSO workshops around the world and also presented CICS-related topics at the WebSphere Conference in Vienna in 2001, at the ITSO EMEA Software Forum in La Hulpe in 2002, at IBM Israel WebSphere Software Symposium Events in 2004 and 2005, and at the ITSO EMEA 2007 zSeries® Software Forum in Amsterdam.

Nigel Williams is a Certified IT Specialist working in the IBM Design Center for On Demand Business in Montpelier. He specializes in core business transformation, connectors, and service-oriented architectures. He is the author of several papers and IBM Redbooks publications and he speaks frequently on CICS and WebSphere topics. Previously, Nigel worked at the Hursley software lab as a software developer, in systems test, and as customer support for the CICS Early Support Program. He holds a degree in Mathematics and Economics from Surrey University.

Thanks to the following people for their contributions to this project:

Bob Haimowitz
International Technical Support Organization, Raleigh Center

Richard Conway
International Technical Support Organization, Raleigh Center

Mark Cocker
IBM Hursley, CICS Technical Strategy and Planning

Dennis Weiand
IBM ATS Dallas

Grant Shayler
IBM Hursley, CICSPlex SM Level 3 Service

Paul Johnson
IBM Hursley, CICS Systems Management Strategy, Planning and Design

Phil Hanson
IBM Hursley, CICS Product Manager

Phil Wakelin
IBM Hursley, CICS Transaction Gateway Technical Planner

Become a published author

Join us for a two- to six-week residency program! Help write an IBM Redbooks publication dealing with specific products or solutions, while getting hands-on experience with leading-edge technologies. You will team with IBM technical professionals, Business Partners, and customers.

Your efforts will help increase product acceptance and customer satisfaction. As a bonus, you will develop a network of contacts in IBM development labs, and increase your productivity and marketability.

Find out more about the residency program, browse the residency index, and apply online at:

ibm.com/redbooks/residencies.html

Comments welcome

Your comments are important to us!

We want our Redbooks to be as helpful as possible. Send us your comments about this or other Redbooks in one of the following ways:

- Use the online **Contact us** review book form found at:

  ibm.com/redbooks

- Send your comments in an e-mail to:

  redbook@us.ibm.com

- Mail your comments to:

  IBM Corporation, International Technical Support Organization
  Dept. HYTD Mail Station P099
  2455 South Road
  Poughkeepsie, NY 12601-5400

Summary of changes

This section describes the technical changes made in this edition of the book and in previous editions. This edition may also include minor corrections and editorial changes that are not identified.

Summary of Changes
for SG24-7144-01
for CICS Web Services Workload Management and Availability
as created or updated on March 28, 2008.

March 2008, Second Edition

This revision reflects the addition, deletion, or modification of new and changed information described below.

New information

- Chapter 4
- Chapter 5
- Chapter 6
- Chapter 7
- Appendix B
- Appendix C
- Appendix D

Changed information

- Chapter 1
- Chapter 2
- Chapter 3

1

Overview of Web services

This chapter focuses on some of the architectural concepts that need to be considered in a Web services project. We define and discuss *service-oriented architecture* (SOA) and the relationship between SOAs and Web services.

We then take a closer look at Web services, a technology that enables you to invoke applications using Internet protocols and standards. The technology is called “Web services” because it integrates services (applications) using Web technologies (the Internet and its standards).

1.1 Introduction

There is a strong trend for companies to integrate existing systems to implement IT support for business processes that cover the entire business cycle. Today, interactions already exist using a variety of schemes that range from very rigid point-to-point electronic data interchange (EDI) interactions to open Web auctions. Many companies have already made some of their IT systems available to all of their divisions and departments, or even their customers or partners on the Web. However, techniques for collaboration vary from one case to another and are thus proprietary solutions; systems often collaborate without any vision or architecture.

Thus, there is an increasing demand for technologies that support the connecting or sharing of resources and data in a very flexible and standardized manner. Because technologies and implementations vary across companies and even within divisions or departments, unified business processes cannot be smoothly supported by technology. Integration has been developed only between units that are already aware of each other and that use the same static applications.

Furthermore, there is a need to further structure large applications into building blocks in order to use well-defined components within different business processes. A shift towards a *service-oriented* approach will not only standardize interaction, but also allows for more flexibility in the process. The complete value chain within a company is divided into small modular functional units, or services. A service-oriented architecture thus has to focus on how services are described and organized to support their dynamic, automated discovery and use.

Companies and their sub-units should be able to easily provide services. Other business units can use these services in order to implement their business processes. This integration can be ideally performed during the runtime of the system, not just at the design time.

1.2 Service-oriented architecture

This section is a short introduction to the key concepts of a service-oriented architecture. The architecture makes no statements about the infrastructure or protocols it uses. Therefore, you can implement a service-oriented architecture using technologies other than Web technologies.

As shown in Figure 1-1, a service-oriented architecture contains three basic components:

- A service provider

  The service provider creates a Web service and possibly publishes to the service broker the information necessary to access and interface with the Web service.

- A service broker

  The service broker (also known as a service registry) makes the Web service access and interface information available to any potential service requester.

- A service requestor

  The service requester binds to the service provider in order to invoke one of its Web services, having optionally placed entries in the broker registry using various find operations.

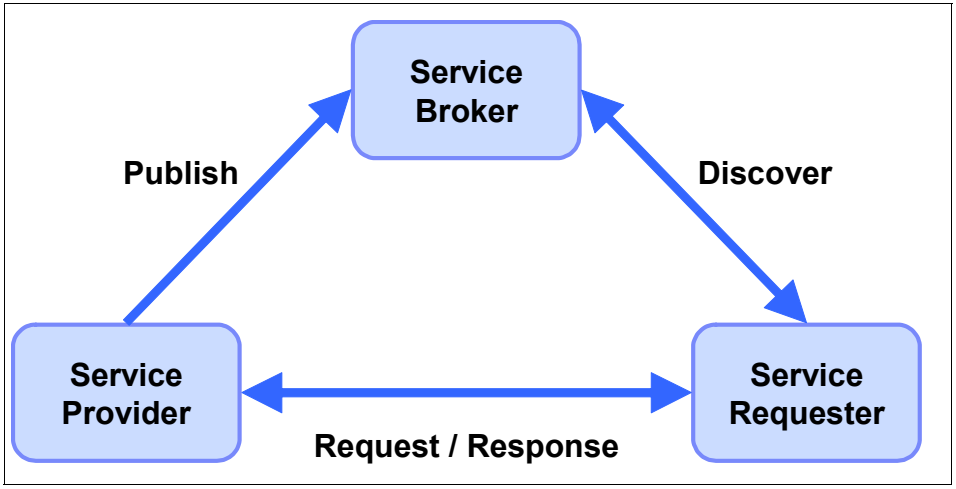


Figure 1-1 Web services components and operations

Each component can also act as one of the two other components. For example, if a service provider needs information that it can only acquire from some other service, it acts as a service requester while still serving the original request.

- The *service provider* creates a Web service and possibly publishes its interface and access information to the service broker.
- The *service broker* (also known as *service registry*) is responsible for making the Web service interface and implementation access information available to any potential service requestor.

- The *service requestor* binds to the service provider in order to invoke one of its Web services, having optionally placed entries in the broker registry using various find operations.

1.2.1 Characteristics

The service-oriented architecture uses a loose coupling between the participants. Such a loose coupling provides greater flexibility as follows:

- Old and new functional blocks are encapsulated into components that work as services.
- Functional components and their interfaces are separated. Therefore, new interfaces can be plugged in more easily.
- Within complex applications, the control of business processes can be isolated. A business rule engine can be incorporated to control the workflow of a defined business process. Depending on the state of the workflow, the engine calls the respective services.

1.2.2 Web services versus service-oriented architectures

The service-oriented architecture has been used under various guises for many years. It can be, and has been, implemented using a number of different distributed computing technologies, such as CORBA or messaging middleware. The effectiveness of service-oriented architectures in the past has always been limited by the ability of the underlying technology to interoperate across the enterprise.

Web services technology is an ideal technology choice for implementing a service-oriented architecture:

- Web services are standards based. Interoperability is a key business advantage within the enterprise and is crucial in B2B scenarios.
- Web services are widely supported across the industry. For the very first time, all major vendors are recognizing and providing support for Web services.
- Web services are platform and language agnostic; there is no bias for or against a particular hardware or software platform. Web services can be implemented in any programming language or toolset. This is important because there will be continued industry support for the development of standards and interoperability between vendor implementations.
- This technology provides a migration path to gradually enable existing business functions as Web services are needed.
- This technology supports synchronous and asynchronous, RPC-based, and complex message-oriented exchange patterns.

Conversely, there are many Web services implementations that are not a service-oriented architecture. For example, the use of Web services to connect two heterogeneous systems directly together is not an SOA. These uses of Web services solve real problems and provide significant value on their own. They may form the starting point of an SOA.

In general, an SOA has to be implemented at an enterprise or organizational level in order to harvest many of the benefits.

For more information about the relationship between Web services and service-oriented architectures, or the application of IBM Patterns for e-business to a Web services project, refer to *Patterns: Service-Oriented Architecture and Web Services*, SG24-6303.

1.3 Web services

Web services perform encapsulated business functions, ranging from simple request-reply to full business process interactions. These services can be new applications or just wrapped around existing business functions to make them network-enabled. Services can rely on other services to achieve their goals.

It is important to note from this definition that a Web service is not constrained to using SOAP over HTTP/S as the transport mechanism. Web services are equally at home in the messaging world.

1.3.1 Properties of a Web service

All Web services share the following properties:

- Web services are self-contained.

  On the client side, no additional software is required. A programming language with XML and HTTP client support is enough to get you started. On the server side, merely an HTTP server and a SOAP server are required.

- Web services are self-describing.

  Using Web Services Description Language (WSDL), all the information required to implement a Web service as a provider, or to invoke a Web service as a requester, is provided.

- Web services can be published, located, and invoked across the Web.

  This technology uses established lightweight Internet standards such as HTTP. It leverages the existing infrastructure.

- Web services are modular.

  Simple Web services can be aggregated to more complex ones, either using workflow techniques or by calling lower-layer Web services from a Web service implementation. Web services can be chained together to perform higher-level business functions. This shortens development time and enables best-of-breed implementations.

- Web services are language-independent and interoperable.

  The client and server can be implemented in different environments. Theoretically, any language can be used to implement Web service clients and servers.

- Web services are inherently open and standard-based.

  XML and HTTP are the major technical foundations for Web services. A large part of the Web service technology has been built using open-source projects. Therefore, vendor independence and interoperability are realistic goals.

- Web services are loosely coupled.

  Traditionally, application design has depended on tight interconnections at both ends. Web services require a simpler level of coordination that allows a more flexible reconfiguration for an integration of the services in question.

- Web services provide programmatic access.

  The approach provides no graphical user interface; it operates at the code level. Service consumers have to know the interfaces to Web services, but do not have to know the implementation details of services.

- Web services provide the ability to wrap existing applications.

Already existing stand-alone applications can easily be integrated into the service-oriented architecture by implementing a Web service as an interface.

1.3.2 Core standards

Web services are built upon four core standards.

Extensible Markup Language (XML)

XML is the foundation of Web services. However, since much information has already been written about XML, we do not describe it in this document. You can find information about XML at:

http://www.w3.org/XML/

SOAP

Originally proposed by Microsoft®, SOAP was designed to be a simple and extensible specification for the exchange of structured, XML-based information in a decentralized, distributed environment. As such, it represents the main means of communication between the three actors in an SOA: the service provider, the service requester, and the service broker. A group of companies, including IBM, submitted SOAP to the W3C for consideration by its XML Protocol Working Group. There are currently two versions of SOAP: Version 1.1 and Version 1.2.

The SOAP 1.1 specification contains three parts:

- An envelope that defines a framework for describing message content and processing instructions. Each SOAP message consists of an envelope that contains an arbitrary number of headers and one body that carries the payload. SOAP messages might contain faults; faults report failures or unexpected conditions.
- A set of encoding rules for expressing instances of application-defined data types.
- A convention for representing remote procedure calls and responses.

A SOAP message is, in principle, independent of the transport protocol that is used, and can, therefore, potentially be used with a variety of protocols, such as HTTP, JMS, SMTP, or FTP. Right now, the most common way of exchanging SOAP messages is through HTTP.

The way SOAP applications communicate when exchanging messages is often referred to as the message exchange pattern (MEP). The communication can be either one-way messaging, where the SOAP message only goes in one direction, or two-way messaging, where the receiver is expected to send back a reply.

Due to the characteristics of SOAP, it does not matter what technology is used to implement the client, as long as the client can issue XML messages. Similarly, the service can be implemented in any language, as long as it can process XML messages.

Note: The authors of the SOAP 1.1 specification declared that the acronym SOAP stands for Simple Object Access Protocol. The authors of the SOAP 1.2 specification decided not to give any meaning to the acronym SOAP.

Web Services Description Language (WSDL)

This standard describes Web services as abstract service endpoints that operate on messages. Both the operations and the messages are defined in an abstract manner, while the actual protocol used to carry the message and the endpoint's address are concrete.

WSDL is not bound to any particular protocol or network service. It can be extended to support many different message formats and network protocols. However, because Web services are mainly implemented using SOAP and HTTP, the corresponding bindings are part of this standard.

The WSDL 1.1 specification only defines bindings that describe how to use WSDL in conjunction with SOAP 1.1, HTTP GET and POST, and MIME. The specification for WSDL 1.1 can be found at:

`http://www.w3.org/TR/wsdl`

WSDL 2.0 provides a model as well as an XML format for describing Web services. It enables you to separate the description of the abstract functionality offered by a service from the concrete details of a service description. It also describes extensions for Message Exchange Patterns, SOAP modules, and a language for describing such concrete details for SOAP1.2 and HTTP.

There are eight Message Exchange Patterns defined. CICS TS V3.2 supports four of them:

- In-Only

  A request message is sent to the Web service provider, but the provider is not allowed to send any type of response to the Web service requester.

- In-Out

  A request message is sent to the Web service provider, and a response message is returned. The response message can be a normal SOAP message or a SOAP fault.

- In-Optional-Out

  A request message is sent to the Web service provider, and a response message is optionally returned to the requester. The response message can be a normal SOAP message or a SOAP fault.

- Robust-In-Only

  A request message is sent to the Web service provider, and no response message is returned to the requester unless an error occurs. In this case, a SOAP fault message is sent to the requester.

The other four Message Exchange Patterns that CICS TS V3.2 does not support are:

- Out-Only
- Robust-Out-Only
- Out-In
- Out-Optional-In

The specification for WSDL 2.0 can be found at:

http://www.w3.org/TR/wsdl20

Universal Description, Discovery, and Integration (UDDI)

The Universal Description, Discovery, and Integration standard defines a means to publish and to discover Web services. As of this writing, UDDI Version 3.0 has been finalized, but UDDI Version 2.0 is still more commonly used. For more information, refer to:

http://www.uddi.org/
http://www.oasis-open.org/specs/index.php#wssv1.0

1.3.3 Web Services Interoperability (WS-I)

Web services can be used to connect computer systems together across organizational boundaries. Therefore, a set of open non-proprietary standards that all Web services adhere to maximizes the ability to connect disparate systems together.

The Web Service Interoperability group (WS-I) is an organization that promotes open interoperabiltity between Web services regardless of platform, operating systems, and programming languages. To promote this cause, the WS-I has released a basic profile that outlines a set of specifications to which WSDL documents and Web services traffic (SOAP over HTTP transport) must adhere in order to be WS-I compliant. The full list of specifications can be found at the WS-I Web site:

http://www.ws-i.org/

IBM is a member of the WS-I community, and CICS support for Web services is fully compliant with the WS-I basic profile 1.0.

1.3.4 Additional standards

There are other Web services specifications that are now supported by CICS. For a list of the limitations of CICS support, refer to *CICS Web Services Guide,* SC34-6838.

Web Services Atomic Transaction

This specification, commonly known as WS-Atomic Transaction, defines the atomic transaction coordination type for transactions of a short duration. Together with the Web Services Coordination specification, it defines protocols for short term transactions that enable transaction processing systems to interoperate in a Web services environment. Transactions that use WS-Atomic Transaction have the properties of atomicity, consistency, isolation, and durability (ACID).

Web Services Security: SOAP Message Security

This specification is a set of enhancements to SOAP messaging that provides message integrity and confidentiality. The specification provides three main mechanisms that can be used independently or together:

- The ability to send security tokens as part of a message, and for associating the security tokens with message content
- The ability to protect the contents of a message from unauthorized and undetected modification (message integrity)
- The ability to protect the contents of a message from unauthorized disclosure (message confidentiality)

Web Services Trust Language

This specification, commonly known as WS-Trust, defines extensions that build on Web Services Security to provide a framework for requesting and issuing security tokens, and broker trust relationships.

SOAP Message Transmission Optimization Mechanism (MTOM)

This specification is one of a related pair of specifications that define how to optimize the transmission and format of a SOAP message. MTOM defines:

- How to optimize the transmission of base64 binary data in SOAP messages.
- How to implement optimized MIME multipart serialization of SOAP messages in a binding, independent way.

- The implementation of MTOM relies on the related XML-binary Optimized Packaging (XOP) specification. As these two specifications are so closely linked, they are normally referred to as MTOM/XOP.

1.4 IBM WebSphere Service Registry and Repository

IBM provides an enterprise strength solution that enables governance of SOA artifacts, most of which are related to Web services. The IBM WebSphere Service Registry and Repository (WSRR) product is such a solution.

The product provides an integrated service metadata repository to govern services and manage the service life cycle, promoting visibility and consistency, and reducing redundancy in your organization. You can seamlessly publish and find capabilities across all phases of SOA, enriching connectivity with dynamic and efficient interactions between services at runtime.

You can use the *CICS TS support for WebSphere Service Registry and Repository* SupportPac™ CA1N to publish Web service artifacts that you develop for CICS in WSRR. You can also fetch them back into the CICS environment. CICS SupportPacs are available at:

`http://www.software.ibm.com/ts/cics/txppacs`

2

CICS implementation of Web services

In this chapter, we discuss how CICS implements Web services. We begin by reviewing the support for SOAP that CICS provided in the SOAP for CICS feature with CICS TS V2. Then we discuss the Web services support of CICS TS V3.1 and the enhancements provided with CICS TS V3.2.

We continue by showing you how to prepare to run a CICS application as a service provider and what happens inside CICS when a service request arrives for a service provider application. Likewise, we discuss preparing to run a CICS application as a service requester and how CICS processes the outbound service request. This leads us to a discussion of the resource definitions that support Web services, namely, the URIMAP, PIPELINE, and WEBSERVICE definitions.

We conclude the chapter by introducing the tools associated with designing and developing Web service applications in CICS.

2.1 SOAP support in CICS TS V2

In 2003, IBM delivered its first support for SOAP in the CICS product when it announced that CICS SupportPac CA1M was available for use with CICS TS V1.3 and CICS TS V2.2. SupportPac CA1M was only intended to be a preview of how CICS might support SOAP. When IBM delivered CICS TS V2.3, it also delivered the SOAP for CICS feature for use in both CICS TS V2.2 and V2.3. Both the SupportPac CA1M and the SOAP for CICS feature implement a pipeline approach to processing SOAP messages.

2.1.1 Pipelines

A *pipeline* is a sequence of programs arranged so that the output from one program is used as input to the next. The SOAP for CICS feature consists of a pipeline that supports both service provider and service requester applications.

A service provider pipeline is a pipeline that consists of user-provided and system-provided programs. They receive an inbound SOAP message, process the contents, and send a response.

A service requester pipeline is a pipeline that consists of user-provided and system-provided programs that send an outbound SOAP message. They receive the response and process the contents of the response.

Figure 2-1 shows a service provider pipeline.

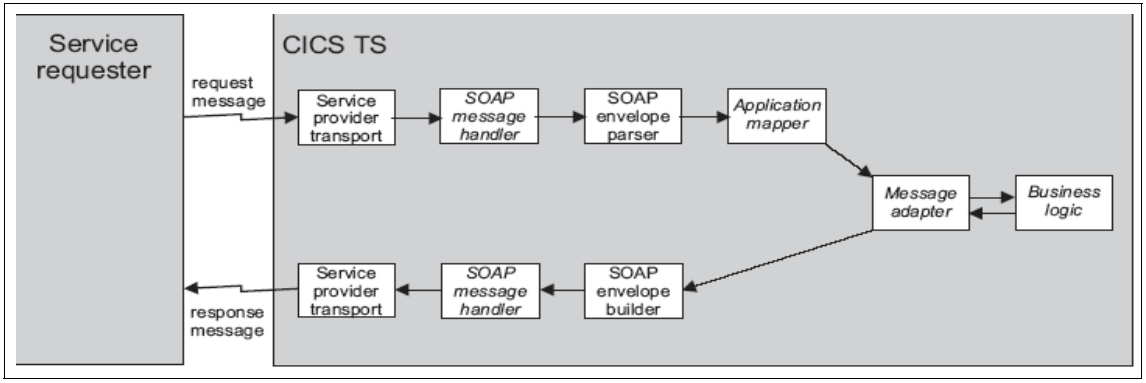


Figure 2-1 Service provider pipeline

The processing stages shown in Figure 2-1 on page 14 are as follows:

- The service provider transport stage (inbound)

  This pipeline stage is responsible for extracting a SOAP request from an incoming message. The SOAP for CICS feature provides a service provider transport program for the HTTP and WebSphere MQ message transports.

- The SOAP message handler (user-written)

  You can use this program to work with the complete SOAP input message. For example, you can extract information from the message or modify its contents.

- The SOAP envelope parser

  This pipeline stage extracts information from the SOAP request envelope and removes the envelope from the incoming message.

- The application mapper (user-written)

  Use this program to specify the name of the message adapter program that is to be invoked for an inbound SOAP request.

- The message adapter (user-written)

  The message adapter is the interface between the pipeline and the business logic. The program:

  a. Parses the SOAP request body

  b. Invokes the business logic (typically, by using an EXEC CICS LINK command with a COMMAREA)

  c. Constructs the SOAP response

- The SOAP envelope builder

  This pipeline stage builds the SOAP response envelope and adds it to the outgoing message.

- The SOAP message handler (user-written)

  You can use this program to work with the complete SOAP output message. For example, you can add SOAP headers to the output message.

- The service provider transport stage (outbound)

  This pipeline stage is responsible for packaging a SOAP response within an outgoing message. The SOAP for CICS feature provides a service provider transport program for the HTTP and WebSphere MQ message transports.

2.1.2 Limitations

The SOAP for CICS feature has the following limitations:

- You can use only one pipeline for service provider applications, and one for service requester applications.
- You may define only one message handler per CICS region.
- XML parsing and generation of the SOAP body must be either user-written, or generated by a tool such as WebSphere Studio Enterprise Edition (WSED), or the more recent WebSphere Developer for System z™.
- It supports only Version 1.1 of the SOAP protocol.
- It does not support the WS-Security and WS-Atomic Transaction specifications.

2.2 Support for Web services in CICS TS V3

Application programs running in CICS TS V3 can participate in a heterogeneous Web services environment as service requesters, service providers, or both, using either an HTTP transport or an MQ transport.

2.2.1 What is provided in CICS TS V3.1

CICS TS V3.1 provides the following capabilities:

- It includes a Web services assistant utility.

  The Web services assistant utility contains two programs: DFHWS2LS and DFHLS2WS.

  - DFHWS2LS helps you map an existing WSDL document into a high level programming language data structure.
  - DFHLS2WS creates a new WSDL document from an existing language structure.

  The Web services assistant supports the following programming languages:

  - COBOL
  - PL/I
  - C
  - C++

- It supports two different approaches to deploying your CICS applications in a Web services environment.
  - You can use the Web services assistant.

    The Web services assistant helps you deploy an application with the least amount of programming effort. For example, if you want to expose an existing application as a Web service, you can start with a high-level language data structure and use DFHLS2WS to generate the Web services description. Alternatively, if you want to communicate with an existing Web service, you can start with its Web service description and use DFHWS2LS to generate a high level language structure that you can use in your program.

    Both DFHLS2WS and DFHWS2LS also generate a file called the wsbind file. When your application runs, CICS uses the wsbind file to transform your application data into a SOAP message on output and to transform the SOAP message to application data on input.
  - You can take complete control of the processing of your data.

    You can write your own code to map between your application data and the message that flows between the service requester and provider. For example, if you want to use non-SOAP messages within the Web service infrastructure, you can write your own code to transform between the message format and the format used by your application.
- It reads a pipeline configuration file created by the CICS system programmer to determine which message handlers should be invoked in a pipeline.

Note: A pipeline can be configured as either a service requester pipeline or a service provider pipeline but not both.

Whether you use the Web services assistant or take complete control of the processing yourself, you can write your own message handlers to perform additional processing on your request and response messages. You can also use CICS-supplied message handlers.

- It supplies message handlers designed especially to help you process SOAP messages.

  The pipelines that CICS uses to process Web service requests and responses are generic, in that there are few restrictions on what processing can be performed in each message handler. However, many Web service applications use SOAP messages, and any processing of those messages should comply with the SOAP specification. Therefore, CICS provides special SOAP message handler programs that can help you to configure your pipeline as a SOAP node.

  - A service requester pipeline is the initial SOAP sender for the request, and the ultimate SOAP receiver for the response.
  - A service provider pipeline is the ultimate SOAP receiver for the request, and the initial SOAP sender for the response.

  You cannot configure a CICS pipeline to function as an intermediary node in a SOAP message path.

  The CICS-provided SOAP message handlers can be configured to invoke one or more user-written header processing programs and to enforce the presence of particular headers in the SOAP message.

- It allows you to configure many different pipelines.

  You can configure a pipeline to support SOAP 1.1 or SOAP 1.2. Within your CICS system, you can have some pipelines that support SOAP 1.1 and others that support SOAP 1.2.

- It provides the following resource definitions to help you configure your support for Web services:

  - PIPELINE
  - URIMAP
  - WEBSERVICE

  If you used the SOAP for CICS feature, you may be able to use CICS resource definitions to replace the logic you provided in your pipeline programs to distinguish one application from another. For example, in a service provider, you may be able to replace code that distinguishes between applications based upon a URI, with a suitable set of URIMAP resources.

- It provides the following EXEC CICS application programming interface (API) commands:

  - SOAPFAULT ADD | CREATE | DELETE
  - INQUIRE WEBSERVICE
  - INVOKE WEBSERVICE

- It conforms to open standards, including:
  - SOAP 1.1 and 1.2
  - HTTP 1.1
  - WSDL 1.1
- It ensures maximum interoperability with other Web services implementations by conforming to the Web Services Interoperability Organization (WS-I) Basic Profile 1.0.
- It supports the WS-Atomic Transaction and WS-Security specifications.

2.2.2 What is new in CICS TS V3.2

CICS TS V3.2 provides the following new functions:

- It supports the WSDL 2.0 specification in addition to WSDL 1.1. CICS support for WSDL 2.0, however, is subject to some restrictions that are documented in the *CICS Web Services Guide*, SC34-6838.

  The Web services assistant utilities, DFHWS2LS and DFHLS2WS, have been enhanced to enable creation of a Web service description that complies with WSDL 2.0. You can create both WSDL 1.1 and WSDL 2.0 documents during the same run of the assistant utility.

  The batch jobs have new and modified parameters that provide you with more flexibility:
  - You can specify an absolute URI for your Web service to DFHLS2WS.
  - DFHWS2LS automatically determines the WSDL version of the Web service description that has been supplied as input.
  - You can select a specific wsdl:Service element within the Web service description.
  - You can specify a subset of wsdl:Operation elements that you want to invoke.

  These enhancements are useful when you have a large WSDL file with many wsdl:Service and wsdl:Operation elements in it.
- In accordance with the WSDL 2.0 specification, CICS now supports four out of the eight Message Exchange Patterns (MEPs). These patterns describe typical interactions between requester and provider. The patterns are:
  - In-Only

    A request message is sent to the Web service provider, but the provider is not allowed to send any type of response to the Web service requester.

- In-Out

  A request message is sent to the Web service provider, and a response message is returned. The response message can be a normal SOAP message or a SOAP fault.

- In-Optional-Out

  A request message is sent to the Web service provider, and a response message is optionally returned to the requester. The response message can be a normal SOAP message, or a SOAP fault.

- Robust In-Only

  A request message is sent to the Web service provider, and no response message is returned to the requester unless an error occurs. In this case, a SOAP fault message is sent to the requester.

- When CICS is acting as a service requester, that is, if your program issues an EXEC CICS INVOKE WEBSERVICE command, you can now define how long CICS should wait for a reply. The PIPELINE resource has a new RESPWAIT attribute that determines how many seconds CICS should wait. If you do not set a value for this attribute, either the default timeout for the transport protocol or the dispatcher timeout for the transaction is used.

  The default timeout for HTTP is 10 seconds; the default timeout for WebSphere MQ is 60 seconds. If the value for the dispatcher timeout for the transaction is less than the default for either protocol, the timeout occurs with the dispatcher.

- New context containers have been provided in the Web service provider and requester pipelines to support WSDL 2.0.

- CICS TS V3.2 supports the SOAP Message Transmission Optimization Mechanism (MTOM) and XML-binary Optimized Packaging (XOP) specification commonly referred to as MTOM/XOP. These specifications define a method for optimizing the transmission of base64Binary data within SOAP messages.

  CICS implements this support in both requester and provider pipelines when the transport protocol is HTTP or HTTPS. You can configure MTOM/XOP support by using additional options in the pipeline configuration file. With this support, base64Binary data is sent as a binary attachment and not directly in the SOAP message.

- CICS support for Web services has been extended to comply with the WSDL 1.1 Binding Extension for SOAP 1.2 specification. This specification defines the binding extensions that are required to indicate that Web service

messages are bound to the SOAP 1.2 protocol. The goal is to provide functionality that is comparable with the binding for SOAP 1.1.

- The support that CICS provides for securing Web services has been enhanced to include an implementation of the Web Services Trust Language (WS-Trust) specification. CICS TS V3.2 can interoperate with a Security Token Service (STS), such as Tivoli® Federated Identity Manager, to validate and issue security tokens in Web services. This enables CICS to send and receive messages that contain a wide variety of security tokens, such as SAML assertions and Kerberos tokens, to interoperate securely with other Web services. CICS support for WS-Trust specification is subject to some restrictions, as shown in the *CICS Web Services Guide*, SC34-6838.

2.2.3 Comparing CICS TS V3 with the SOAP for CICS feature

Table 2-1 summarizes some of the differences between the support for Web services in CICS TS V3.1 and CICS TS V3.2, and the SOAP support in the SOAP for CICS feature.

Table 2-1 Comparison of CICS TS V3 Web services with SOAP for CICS feature

Description	CICS TS V3.2	CICS TS V3.1	SOAP for CICS feature
Pipeline data passing mechanism	Channels and containers	Channels and containers	BTS containers
Number of pipelines	Multiple per CICS region	Multiple per CICS region	One service requester pipeline per CICS region One service provider pipeline per CICS region
Number of message handlers	Multiple per each Web service definition	Multiple per each Web service definition	One per CICS region

Description	CICS TS V3.2	CICS TS V3.1	SOAP for CICS feature
Pipeline container names	► DFHWS-APPHANDLER ► DFHWS-BODY ► DFHWS-DATA ► DFHWS-OPERATION ► DFHWS-PIPELINE ► DFHWS-SOAPACTION ► DFHWS-SOAPLEVEL ► DFHWS-TRANID ► DFHWS-URI ► DFHWS-USERID ► DFHWS-WEBSERVICE ► DFHWS-XMLNS ► DFHERROR ► DFHFUNCTION ► DFHHEADER ► DFHNORESPONSE ► DFHREQUEST ► DFHRESPONSE ► DFH-HANDLERPLIST ► DFH-SERVICEPLIST ► DFHWS-MEP ► DFHWS-CID-DOMAIN ► DFHWS-MTOM-IN ► DFHWS-MTOM-OUT ► DFHWS-XOP-IN ► DFHWS-XOP-OUT ► DFHWS-IDTOKEN ► DFHWS-RESTOKEN ► DFHWS-SERVICEURI ► DFHWS-STSACTION ► DFHWS-STSFAULT ► DFHWS-STSURI ► DFHWS-TOKENTYPE	► DFHWS-APPHANDLER ► DFHWS-BODY ► DFHWS-DATA ► DFHWS-OPERATION ► DFHWS-PIPELINE ► DFHWS-SOAPACTION ► DFHWS-SOAPLEVEL ► DFHWS-TRANID ► DFHWS-URI ► DFHWS-USERID ► DFHWS-WEBSERVICE ► DFHWS-XMLNS ► DFHERROR ► DFHFUNCTION ► DFHHEADER ► DFHNORESPONSE ► DFHREQUEST ► DFHRESPONSE ► DFH-HANDLERPLIST ► DFH-SERVICEPLIST	► APP-HANDLER ► APP-NAMESPACES ► FAULT ► INPUT ► NAMESPACES ► OUTPUT ► PIPELINE-ERROR ► REQUEST-BODY ► RESPONSE-BODY ► SOAP-ACTION ► TARGET-TRANID ► TARGET-URI ► TARGET-USERID ► USER-CONTAINERS
SOAP protocol level	SOAP 1.1 and 1.2	SOAP 1.1 and 1.2	SOAP 1.1
CICS resource definitions	► PIPELINE ► URIMAP ► WEBSERVICE	► PIPELINE ► URIMAP ► WEBSERVICE	None

Description	CICS TS V3.2	CICS TS V3.1	SOAP for CICS feature
CICS API and SPI	► CREATE PIPELINE ► CREATE URIMAP ► CREATE WEBSERVICE ► INQUIRE WEBSERVICE ► INVOKE WEBSERVICE ► PERFORM PIPELINE SCAN ► SET WEBSERVICE ► SOAPFAULT ADD ► SOAPFAULT CREATE ► SOAPFAULT DELETE	► CREATE PIPELINE ► CREATE URIMAP ► CREATE WEBSERVICE ► INQUIRE WEBSERVICE ► INVOKE WEBSERVICE ► PERFORM PIPELINE SCAN ► SET WEBSERVICE ► SOAPFAULT ADD ► SOAPFAULT CREATE ► SOAPFAULT DELETE	none
XML parsing	CICS wsbind file generated by either CICS Web service assistant or WebSphere Developer for System z	CICS wsbind file generated by either CICS Web service assistant or WebSphere Developer for System z	WebSphere Developer for System z WebSphere Studio Enterprise Developer (WSED) Write your own using Enterprise COBOL
WSDL support	1.1 and 2.0	1.1	1.1
WS-Atomic support	Yes	Yes	None
WS-Security support	Yes	Yes	None
MTOM/XOP support	Yes	none	None
WS-Trust support	Yes	None	None

2.3 CICS as a service provider

When CICS is a service provider, it receives a service request, which is passed through a pipeline to a target application program. The response from the application is returned to the service requester through the same pipeline. In this section, we first discuss how to prepare for running a CICS application as a service provider. Then we discuss how CICS processes the incoming service request.

2.3.1 Preparing to run a CICS application as a service provider

Suppose that we have an existing CICS application that we wish to expose as a Web service that uses the HTTP transport. Suppose also that we wish to use the Web services assistant rather than taking control of the processing ourselves.

In this section, we describe a technique that is suitable for a development environment. We allow CICS to autoinstall the URIMAP and WEBSERVICE definitions by scanning the pickup directory of the pipeline. In a production environment, you may prefer to keep tighter control on your URIMAP and WEBSERVICE resource definitions and use hardcoded definitions, as we describe in 2.6.1, “URIMAP” on page 35 and 2.6.3, “WEBSERVICE” on page 43.

We go through the following steps:

1. We generate the wsbind and WSDL files (application developer).
   a. We first create an HFS directory in which to store the generated files. For example, we might create a directory named /u/SharedProjectDirectory/MyFirstWebServiceProvider.
   b. We next run the DFHLS2WS program. The input we provide to the program includes the following information:
      - The names of the partitioned data set members that contain the high level language structures that the application program uses to describe the Web service request and the Web service response.
      - The fully qualified HFS names of the wsbind file and of the file into which the Web service description is to be written (the WSDL file).
      - The relative URI that a client will use to access the Web service.
      - How CICS should pass data to the target application program (COMMAREA or container).

   Typically, an application developer would perform this step.

2. Create a TCPIPSERVICE resource definition (system programmer).

   The resource definition should specify PROTOCOL(HTTP) and supply information about the port on which inbound requests are received.

   Typically, a system programmer would perform this step.

3. Create a PIPELINE resource definition (system programmer).

   a. Create a service provider pipeline configuration file.

      A pipeline configuration file is an XML file that describes, among other things, the message handler programs and the SOAP header processing programs that CICS will invoke when it processes the pipeline.

   b. Create an HFS directory in which to store installable wsbind and WSDL files.

      We call this directory the “pickup” directory since CICS will pick up the wsbind and WSDL files from this directory and store them on a “shelf” directory.

   c. Create an HFS directory for CICS in which to store installed wsbind files.

      We call this directory the “shelf” directory.

   d. Create a PIPELINE resource definition to handle the Web service request.

      - Specify the CONFIGFILE attribute to point to the file created in step 3a.
      - Specify the WSDIR attribute to point to the directory created in step 3b.
      - Specify the SHELF attribute to point to the directory created in step 3c.

   e. Copy the wsbind and WSDL files created in step 1 to the pickup directory created in step 3b.

4. Install the TCPIPSERVICE and PIPELINE resource definitions (system programmer).

   When the CICS system programmer installs the PIPELINE definition, CICS scans the pickup directory for wsbind files. When CICS finds the wsbind file created in step 1, CICS dynamically creates and installs a WEBSERVICE resource definition for it. CICS derives the name of the WEBSERVICE definition from the name of the wsbind file. The WEBSERVICE definition identifies the name of the associated PIPELINE definition and points to the location of the wsbind file in the HFS.

   During the installation of the WEBSERVICE resource:

   – CICS dynamically creates and installs a URIMAP resource definition. CICS bases the definition on the URI specified in the input to DFHLS2WS (see step 1) and stored by DFHLS2WS in the wsbind file.

– CICS uses the wsbind file to create main storage control blocks to map the inbound service request (XML) to a COMMAREA or a container and to map to XML the outbound COMMAREA or container that contains the response data.

5. Publish WSDL to clients.

   a. Customize the location attribute on the <address> element in the WSDL file so that its value specifies the TCP/IP server name of the machine hosting the service and the port number defined in step 2.

   b. Publish the WSDL to any parties wishing to create clients to this Web service.

2.3.2 Processing the inbound service request

Figure 2-2 shows the processing that occurs when a service requester sends a SOAP message over HTTP to a service provider application running in a CICS TS V3 region.

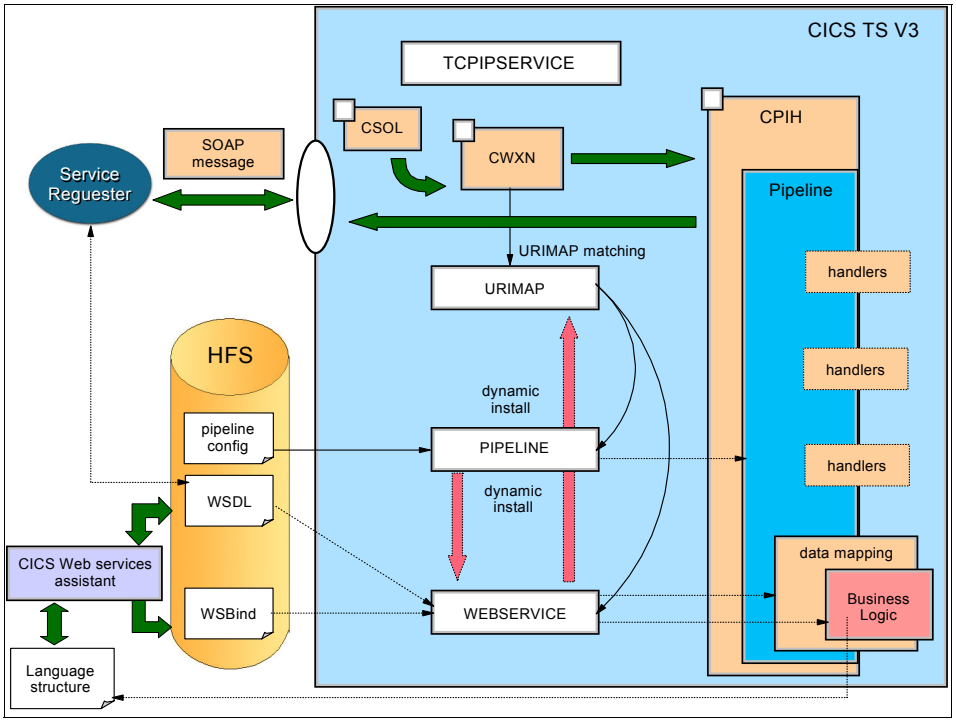


Figure 2-2 Web service runtime service provider processing

The CICS-supplied sockets listener transaction (CSOL) monitors the port specified in the TCPIPSERVICE resource definition for incoming HTTP requests. When the SOAP message arrives, CSOL attaches the transaction specified in the TRANSACTION attribute of the TCPIPSERVICE definition; by default, this will be the CICS-supplied Web attach transaction CWXN.

CWXN finds the URI in the HTTP request and then scans the URIMAP resource definitions for a URIMAP that has its USAGE attribute set to PIPELINE and its PATH attribute set to the URI found in the HTTP request. If CWXN finds such a URIMAP, its uses the PIPELINE and WEBSERVICE attributes of the URIMAP definition to get the name of the PIPELINE and WEBSERVICE definitions that it will use to process the incoming request. CWXN also uses the TRANSACTION attribute of the URIMAP definition to determine the name of the transaction that it should attach to process the pipeline; by default, this will be the CPIH transaction.

CPIH starts the pipeline processing. It uses the PIPELINE definition to find the name of the pipeline configuration file. CPIH uses the pipeline configuration file to determine which message handler programs and SOAP header processing programs to invoke.

A message handler in the pipeline (typically, a CICS-supplied SOAP message handler) removes the SOAP envelope from the inbound request and passes the SOAP body to the data mapper function.

CICS uses the DFHWS-WEBSERVICE container to pass the name of the required WEBSERVICE definition to the data mapper. The data mapper uses the WEBSERVICE definition to locate the main storage control blocks that it needs to map the inbound service request (XML) to a COMMAREA or a container.

The data mapper links to the target service provider application program, providing it with input in the format that it expects. The application program is not aware that it is being executed as a Web service. The program performs its normal processing and then returns an output COMMAREA or container to the data mapper.

The output data from the CICS application program cannot just be sent back to the pipeline code. The data mapper must first convert the output from the COMMAREA or container format into a SOAP body.

2.4 CICS as a service requester

When CICS is a service requester, an application program sends a request, which is passed through a pipeline, to a target service provider. The response from the service provider is returned to the application program through the same pipeline. In this section, we first discuss how to prepare to run a CICS application as a service requester. Then we discuss how CICS processes the outbound service request.

2.4.1 Preparing to run a CICS application as a service requester

Suppose we wish to write a new CICS application that will invoke a Web service. Suppose also that we wish to use the Web services assistant rather than taking control of the processing ourselves.

In this section, we describe a technique that is suitable for a development environment, that is, we let CICS autoinstall WEBSERVICE definitions by scanning the pickup directory of the pipeline. In a production environment, you may prefer to keep tighter control of your WEBSERVICE resource definitions and use hardcoded definitions, as we describe in 2.6.3, “WEBSERVICE” on page 43.

We go through the following steps:

1. Generate the wsbind file and the language structures (application developer).
   a. We first create an HFS directory in which to store the wsbind file. For example, we might create a directory named /u/SharedProjectDirectory/MyFirstWebServiceRequester.
   b. We next run the DFHWS2LS program. The input we provide to the program includes the following information:
      - The fully qualified HFS name of the WSDL file that describes the Web service that we want to request.
      - The names of the partitioned data set members into which DFHWS2LS should put the high level language structures that it generates. The application program uses the language structures to describe the Web service request and the Web service response.
2. Create a PIPELINE resource definition (system programmer).
   a. Create a service requester pipeline configuration file.

      A pipeline configuration file is an XML file that describes, among other things, the message handler programs and the SOAP header processing programs that CICS will invoke when it processes the pipeline.

b. Create an HFS directory in which to store installable wsbind files.

   We call this directory the “pickup” directory, since CICS will pick up the wsbind file from this directory and store it on a “shelf” directory.

c. Create an HFS directory for CICS in which to store installed wsbind files.

   We call this directory the “shelf” directory.

d. Create a PIPELINE resource definition to handle the Web service request.

   - Specify the CONFIGFILE attribute to point to the file created in step 2a.
   - Specify the WSDIR attribute to point to the directory created in step 2b.
   - Specify the SHELF attribute to point to the directory created in step 2c.

e. Copy the wsbind file created in step 1to the pickup directory created in step 2b.

3. Install the PIPELINE resource definition (system programmer).

   When the CICS system programmer installs the PIPELINE definition, CICS scans the pickup directory for wsbind files. When CICS finds the wsbind file created in step 1, CICS dynamically creates and installs a WEBSERVICE resource definition for it. CICS derives the name of the WEBSERVICE definition from the name of the wsbind file. The WEBSERVICE definition identifies the name of the associated PIPELINE definition and points to the location of the wsbind file in the HFS.

   During the installation of the WEBSERVICE resource, CICS uses the wsbind file to create main storage control blocks to map the outbound service request to an XML document and to map the inbound XML response document to a language structure.

4. Use the language structure generated in step 1 to write the application program (application developer).

   a. It issues the following command to place the outbound data into container DFHWS-DATA:

      ```
      EXEC CICS PUT CONTAINER(DFHWS-DATA) CHANNEL(name_of_channel)
      FROM(data_area)
      ```

   b. It issues the following command to invoked the Web service:

      ```
      EXEC CICS INVOKE WEBSERVICE(name_of_WEBSERVICE_definition)
      CHANNEL(name_of_channel) OPERATION(name_of_operation)
      ```

2.4.2 Processing the outbound service request

Figure 2-3 shows the processing that occurs when a service requester running in a CICS TS V3 region sends a SOAP message to a service provider.

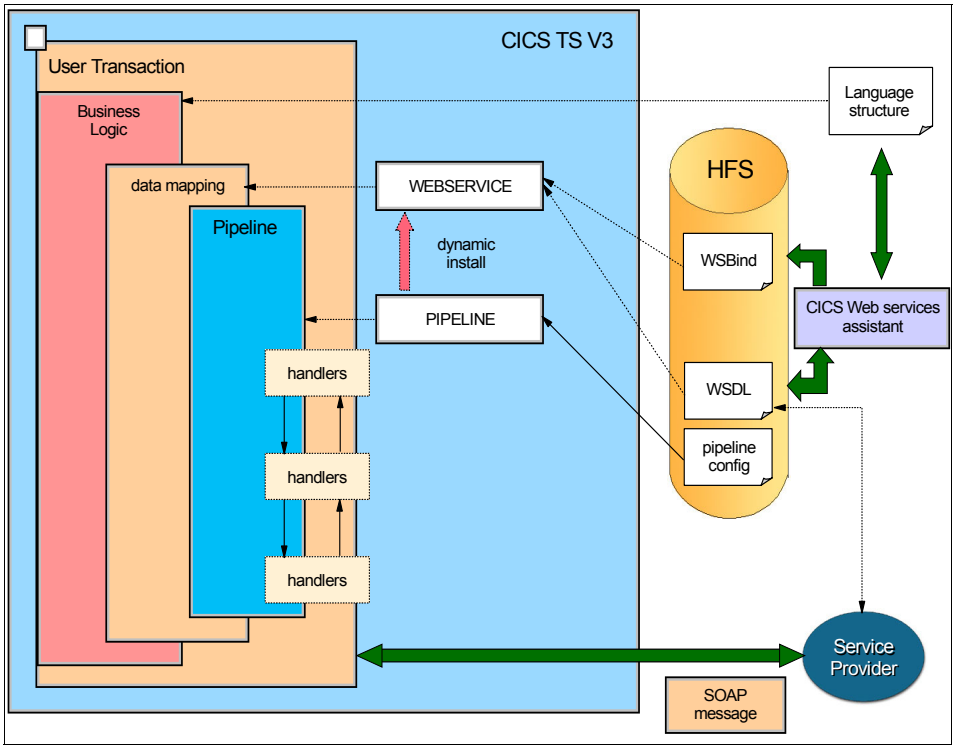


Figure 2-3 Web service requestor resources

When the service requester issues the EXEC CICS INVOKE WEBSERVICE command, CICS uses the information found in the wsbind file that is associated with the specified WEBSERVICE definition to convert the language structure into an XML document. CICS then invokes the message handlers specified in the pipeline configuration file, and they convert the XML document into a SOAP message.

CICS will send the request SOAP message to the remote service provider either through HTTP or WebSphere MQ.

When the response SOAP message is received, CICS will pass it back through the pipeline. The message handlers will extract the SOAP body from the SOAP envelope, and the data mapping function will convert the XML in the SOAP body into a language structure that is passed to the application program in container DFHWS-DATA.

When CICS TS V3.2 is acting as a service requester, that is, if your program issues an INVOKE WEBSERVICE command, you can now define how long CICS waits for a reply. The PIPELINE resource has a new RESPWAIT attribute that determines how many seconds CICS waits. If you do not set a value for this attribute, either the default timeout for the transport protocol or the dispatcher timeout for the transaction is used.

The default timeout for HTTP is 10 seconds; the default timeout for WebSphere MQ is 60 seconds. If the value for the dispatcher timeout for the transaction is less than the default for either protocol, the timeout occurs with the dispatcher.

Figure 2-4 shows a CICS TS V3.2 region acting as both a service provider and service requester.

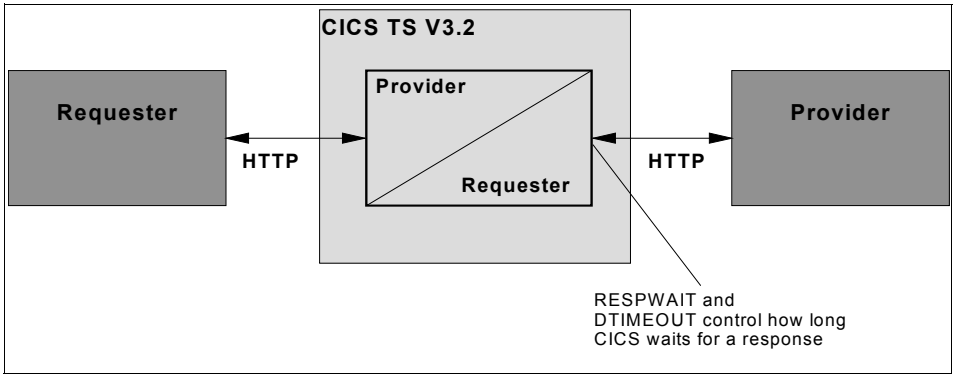


Figure 2-4 CICS TS V3.2 region as a service provider and a service requester

2.5 Catalog manager example application

The CICS catalog manager example application is a COBOL application that is designed to illustrate best practices when connecting CICS applications to external clients and servers.

The example is constructed around a simple sales catalog and order processing application, in which the user can perform these functions:

- List the items in a catalog.
- Inquire on individual items in the catalog.
- Order items from the catalog.

The base application has a 3270 interface. Figure 2-5 shows the structure of the base application.

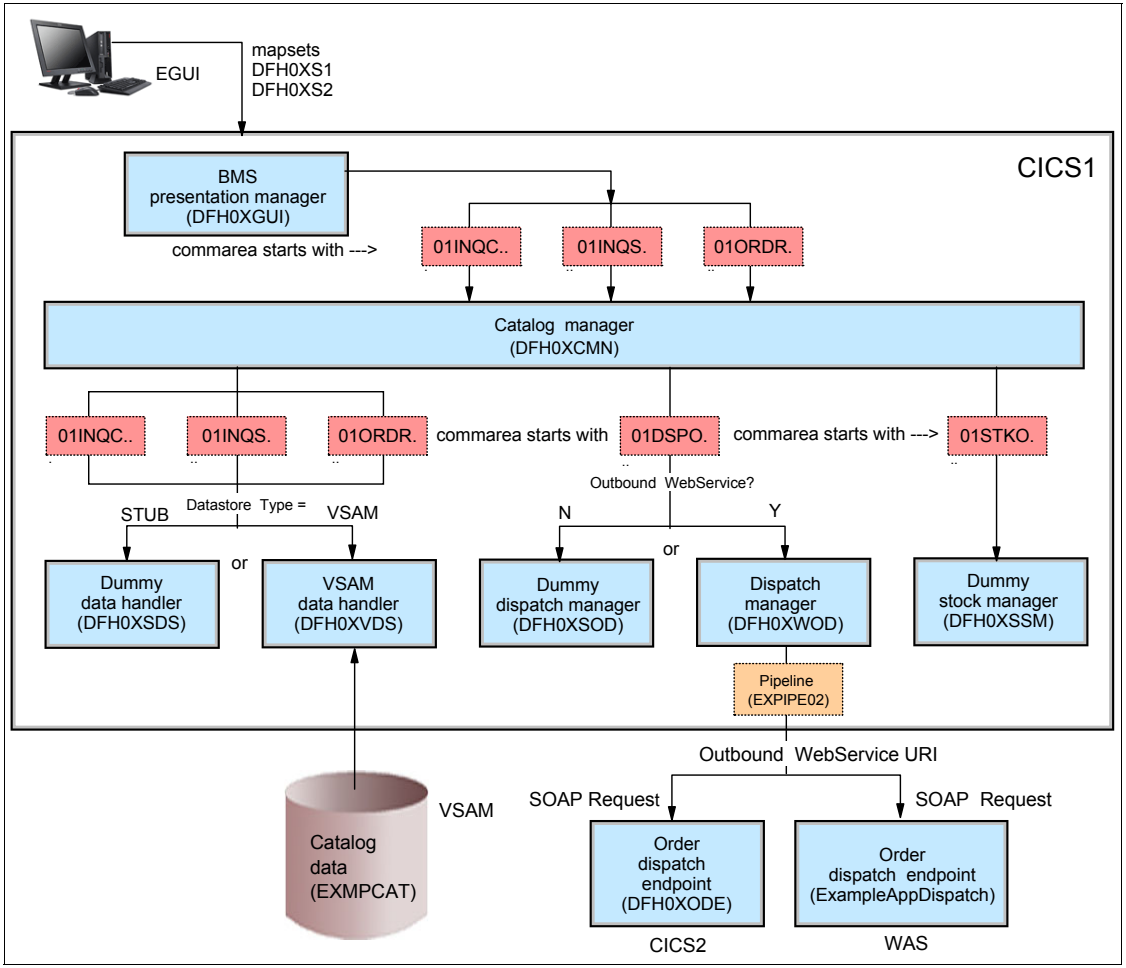


Figure 2-5 Basic catalog manager application structure

The business logic can also be invoked as a Web service in three different ways:

1. By a CICS Web services client program invoked from a 3270 screen
2. By a WebSphere Application Server client application invoked by a browser with direct invocation of the Catalog manager program DFH0XCMN
3. By a WebSphere Application Server application invoked by a browser with three different front-end programs that serve as wrappers for DFH0XCMN

Figure 2-6 shows the structure of the Web service provider application.

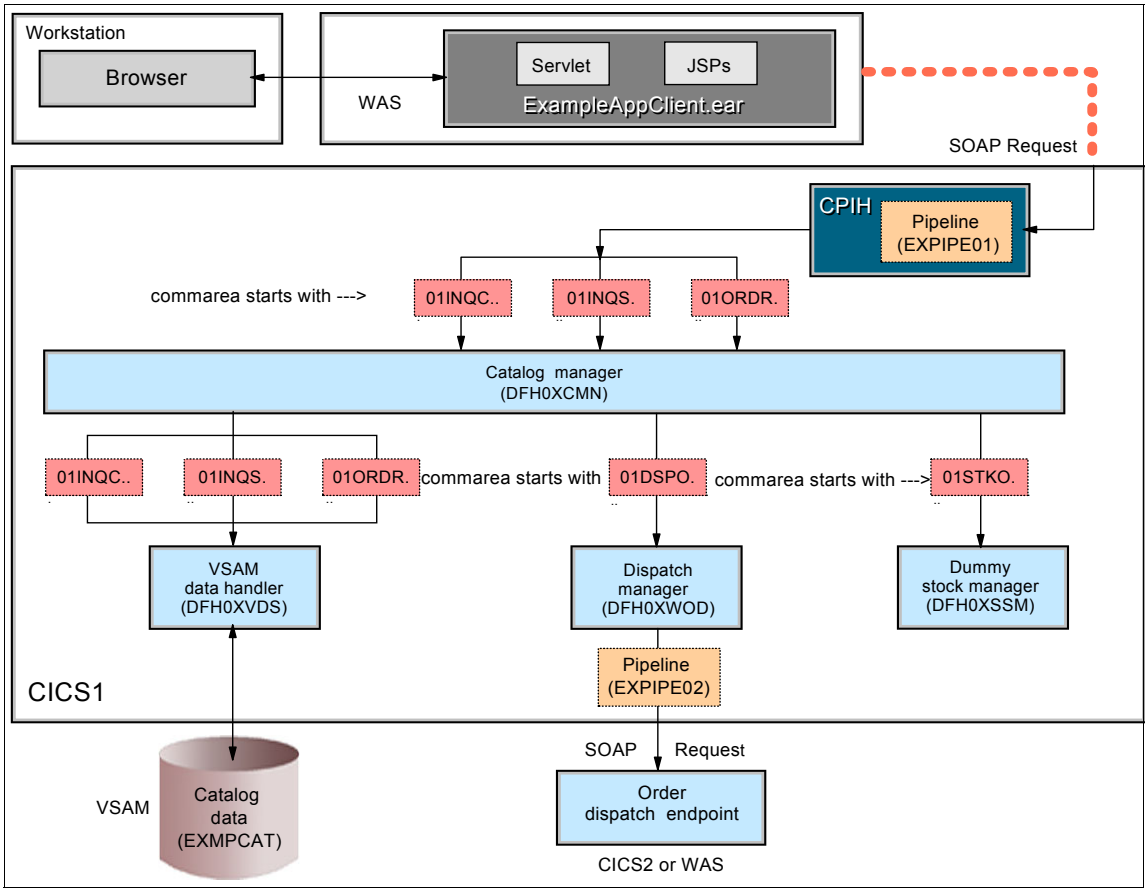


Figure 2-6 Structure of Web service provider application

You configure the sample application with an ECFG transaction. Example 2-1 shows a configuration screen of the ECFG transaction.

Example 2-1 Catalog application configuration screen

```
CONFIGURE CICS EXAMPLE CATALOG APPLICATION

           Datastore Type ==> VSAM               STUB|VSAM
     Outbound WebService? ==> NO                 YES|NO
          Catalog Manager ==> DFHOXCMN
          Data Store Stub ==> DFHOXSDS
          Data Store VSAM ==> DFHOXVDS
      Order Dispatch Stub ==> DFHOXSOD
Order Dispatch WebService ==> DFHOXWOD
            Stock Manager ==> DFHOXSSM
           VSAM File Name ==> EXMPCAT
  Server Address and Port ==>
  Outbound WebService URI ==> http://9.42.170.163:9080/exampleApp/services
                          ==> /dispatchOrderPort
                          ==>
                          ==>
                          ==>
                          ==>
APPLICATION CONFIGURATION UPDATED
PF             3 END                                                 12 CNCL
```

For the complete description of the catalog manager application and its setup, refer to the *CICS Web Services Guide*, SC34-6838.

In 5.1.3, “The CICS configuration” on page 105, we explain how we deployed the catalog manager service application as a service provider application in a CICSplex configuration.

In 6.1, “Configuration update” on page 162, we explain how we deployed the catalog manager application as a service requester application in a CICSplex configuration.

2.6 CICS TS V3 Web service resource definitions

In this section, we provide Web service resource definitions for CICS TS V3.

2.6.1 URIMAP

URIMAP definitions are used to provide three different Web-related facilities in CICS. It is the value of the USAGE attribute on a URIMAP definition that determines which of the three facilities that particular definition provides:

1. Requests from a Web client, to CICS as an HTTP server:

   URIMAP definitions for requests for CICS as an HTTP server have a USAGE attribute of SERVER. These URIMAP definitions match the URLs of HTTP requests that CICS expects to receive from a Web client, and they define how CICS should provide a response to each request. You can use a URIMAP definition to tell CICS to:

   – Provide a *static* response to the HTTP request, using a document template or z/OS UNIX® System Services HFS file.

   – Provide a *dynamic* response to the HTTP request, using an application program that issues EXEC CICS WEB application programming interface commands.

   – Redirect the request to another server, either temporarily or permanently.

   For CICS as an HTTP server, URIMAP definitions incorporate most of the functions that were previously provided by the analyzer program specified on the TCPIPSERVICE definition. An analyzer program may still be involved in the processing path if required.

2. Requests to a server, from CICS as an HTTP client:

   URIMAP definitions for requests from CICS as an HTTP client have a USAGE attribute of CLIENT. These URIMAP definitions specify URLs that are used when a user application, acting as a Web client, makes a request through CICS Web support to an HTTP server. Setting up a URIMAP definition for this purpose means that you can avoid identifying the URL in your application program.

3. Web service requests:

   URIMAP definitions for Web service requests have a USAGE attribute of PIPELINE. These URIMAP definitions associate a URI for an inbound Web service request (that is, a request by which a client invokes a Web service in CICS) with a PIPELINE or WEBSERVICE resource that specifies the processing to be performed. They may also be used to specify:

   – The name of the transaction that CICS should use to run the pipeline.

   – The user ID under which the pipeline transaction runs.

   Figure 2-2 on page 26 illustrates the purpose of a URIMAP definition for mapping Web service requests. In this book, we only concern ourselves with this third use of the URIMAP definition.

You may create URIMAP resource definitions in the following ways:

- Use the CEDA transaction.
- Use the DFHCSDUP batch utility.
- Use CICSPlex SM Business Application Services.
- Use the EXEC CICS CREATE URIMAP command.

When you install a PIPELINE resource, or when you issue a PERFORM PIPELINE SCAN command (using CEMT or the CICS system programming interface), CICS scans the directory specified in the PIPELINE's WSDIR attribute (the pickup directory) and creates URIMAP and WEBSERVICE resources dynamically. For each Web service binding file in the directory, that is, for each file with the wsbind suffix, CICS installs a WEBSERVICE and a URIMAP if one does not already exist. Existing resources are replaced if the information in the binding file is more recent than the existing resources.

Whether you install URIMAP resource definitions dynamically or whether you define them manually, it is possible to change the transaction ID of the pipeline processing transaction.

2.6.2 PIPELINE

A PIPELINE resource definition provides information about the message handlers that will act on a service request and on the response. The information about the message handlers is supplied indirectly; the PIPELINE definition specifies the name of an HFS file, called the pipeline configuration file, which contains an XML description of the message handlers and their configuration.

Figure 2-2 on page 26 and Figure 2-3 on page 30 illustrate the purpose of the PIPELINE resource definition for service provider and service requester pipelines, respectively.

The most important attributes of the PIPELINE definition are as follows:

- WSDIR

  The WSDIR attribute specifies the name of the Web service binding directory (also known as the pickup directory). The Web service binding directory contains Web service binding files that are associated with the PIPELINE, and that are to be installed automatically by the CICS scanning mechanism. When the PIPELINE definition is installed, CICS scans the directory and automatically installs any Web service binding files it finds there.

  If you specify a value for the WSDIR attribute, it must refer to a valid HFS directory to which the CICS region has at least read access. If this is not the case, any attempt to install the PIPELINE resource will fail.

  If you do not specify a value for WSDIR, no automatic scan takes place upon installation of the PIPELINE, and PERFORM PIPELINE SCAN commands will fail.

- SHELF

  The SHELF attribute specifies the name of an HFS directory where CICS will copy information about installed Web services. CICS regions into which the PIPELINE definition is installed must have full permission to the shelf directory: read, write, and the ability to create subdirectories.

  A single shelf directory may be shared by multiple CICS regions and by multiple PIPELINE definitions. Within a shelf directory, each CICS region uses a separate subdirectory to keep its files separate from those of other CICS regions. Within each region's directory, each PIPELINE uses a separate subdirectory.

  After a CICS region performs a cold or initial start, its deletes its subdirectories from the shelf before trying to use the shelf.

- CONFIGFILE

  This attribute specifies the name of the PIPELINE configuration file.

- RESPWAIT

  This attribute determines how many seconds CICS should wait.If you do not set a value for this attribute, either the default timeout for the transport protocol or the dispatcher timeout for the transaction is being used.

  The default timeout for HTTP is 10 seconds; the default timeout for WebSphere MQ is 60 seconds. If the value for the dispatcher timeout for the transaction is less than the default for either protocol, the timeout occurs with the dispatcher.

  Note: The RESPWAIT attribute only applies to service requester pipelines.

Pipeline configuration file

When CICS processes a Web service request, it uses a pipeline of one or more message handlers to handle the request. The configuration of the pipeline will depend upon the needs of the application. The configuration of a pipeline used to handle a Web service request is specified in an XML document, known as a pipeline configuration file. Use a suitable XML editor or text editor to work with your pipeline configuration files.

There are two kinds of pipeline configuration files: one describes the configuration of a service provider pipeline and the other describes the configuration of a service requester pipeline. An example configuration file for a CICS service provider pipeline is shown in Example 2-2.

Example 2-2 Configuration file

```
<?xml version="1.0" encoding="EBCDIC-CP-US"?>
<provider_pipeline xmlns="http://www.ibm.com/software/htp/cics/pipeline"
      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xsi:schemaLocation="http://www.ibm.com/software/htp/cics/pipeline provider.xsd ">
  <service>
    <service_handler_list>
      <handler>
        <program>SETTRNID</program>
        <handler_parameter_list/>
      </handler>
    </service_handler_list>
    <terminal_handler>
      <cics_soap_1.1_handler/>
    </terminal_handler>
  </service>
  <apphandler>DFHPITP</apphandler>
</provider_pipeline>
```

The first line of the pipeline configuration file specifies the XML version and the encoding that is being used by this pipeline. Notice that the CICS-provided basicsoap11provider.xml and basicsoap11requester.xml files specify encoding="EBCDIC-CP-US"?. This setting allows you to test the CICS-provided example catalog management application using the SOAP client that runs inside CICS. If you want to use the WebSphere Application Server client, you have to change the encoding in basicsoap11provider.xml to "UTF-8"?. Similarly, if you want to test the outbound option of the sample application, change the encoding in basicsoap11requester.xml to "UTF-8"?. Refer to Chapter 6, "Configuring CICS as a service requester" on page 161 for a description of how we tested this option.

Each kind of pipeline configuration file is defined by its own schema, and each has a different root element. The root element for a provider pipeline is <provider_pipeline>, while the root element for a requester pipeline is <requester_pipeline>.

The immediate child elements of the <provider_pipeline> element are:

- A mandatory <service> element, which specifies the message handlers that are invoked for every request, including the terminal message handler. The terminal message handler is the last handler in the pipeline.

  Optional message handlers (see “Message handlers” on page 40) are defined in the <handler> tag within the <service_handler_list> tag. The pipeline configuration file in Example 2-2 on page 38 defines a message handler program SETTRNID, which will be invoked before the SOAP 1.1 terminal node, and thus before invoking the data conversion program (DFHPITP).

- An optional <transport> element, which specifies message handlers that are selected at runtime, based upon the resources that are being used for the message transport. For example, for the HTTP transport, you may specify that CICS should invoke the message handler only when the port on which the request was received is defined on a specific TCPIPSERVICE definition. For the WebSphere MQ transport, you may specify that CICS should invoke the message handler only when the inbound message arrives at a specific message queue.

- An optional <apphandler> element, which specifies the name of the program that the terminal message handler will link to by default, that is, the name of the target application program (or wrapper program) that provides the service. Message handlers can specify a different program at runtime by using the DFHWS-APPHANDLER container, so the name coded here is not always the program to which it is linked.

  When you use DFHLS2WS or DFHWS2LS to deploy your service provider, you must specify DFHPITP as the target program. DFHPITP will obtain the name of your target application program (or wrapper program) from the wsbind file.

  The <apphandler> element is used when the last message handler in the pipeline (the terminal handler) is one of the CICS-supplied SOAP message handlers,

  If you do not code an <apphandler> element, one of the message handlers must use the DFHWS-APPHANDLER container to specify the name of the program.

- An optional <service_parameter_list> element, which contains parameters that CICS will make available to the message handlers in the pipeline through container DFH-SERVICEPLIST.

The immediate sub-elements of a <requester_pipeline> element are:

- An optional <service> element, which specifies the message handlers that are invoked for every request
- An optional <transport> element, which specifies message handlers that are selected at runtime, based upon the resources that are being used for the message transport
- An optional <service_parameter_list> element, which contains parameters that CICS will make available to the message handlers in the pipeline through container DFH-SERVICEPLIST

Message handlers

A message handler is a program that is used to process a Web service request during input, and to process the response during output. Message handlers use channels and containers to interact with one another, and with the system.

The message handler interface lets you perform the following tasks in a message handler program:

- Examine the contents of an XML request or response without changing it.

  For example, a message handler that performs logging will examine a message, and copy the relevant information from that message to the log. The message that is passed to the next handler is unchanged.
- Change the contents of an XML request or response.

  For example, a message handler that performs encryption and decryption will receive an encrypted message on input, and pass the decrypted message to the next handler. On output, it will do the opposite: receive a plain text message, and pass an encrypted version to the following handler.
- In a non-terminal message handler, pass an XML request or response to the next message handler in the pipeline.
- In a terminal message handler, call an application program, and generate a response.
- In the request phase of the pipeline, force a transition to the response phase, by absorbing the request, and generating a response.
- Handle errors.

All programs that are used as message handlers are invoked with the same channel interface. The channel holds a number of containers. The containers can be categorized as:

- Control containers

  These are essential to the operation of the pipeline. Message handlers can use the control containers to modify the sequence in which the message handlers are processed.

- Context containers

  In some situations, message handler programs need information about the context in which they are invoked. CICS provides this information in a set of context containers that are passed to the programs. Some of the context containers hold information that you can change in your message handler. For example, in a service provider pipeline, you can change the user ID and transaction ID of the target application program by modifying the contents of the appropriate context containers DFHWS-USERID and DFHWS-TRANID.

  The effect of changing the transaction ID in a message handler is that the data mapping and target application program runs in a separate task using the changed transaction ID. An example of a message handler that changes the transaction ID is provided by *Implementing CICS Web Services*, SG24-7206.

  Setting different transaction IDs based on the name of the Web service, for example, can be useful for accounting and security purposes.

- User containers

  These contain information that one message handler needs to pass to another. The use of user containers is entirely a matter for the message handlers.

SOAP message handlers

The SOAP message handlers are CICS-provided message handlers that you can include in your pipeline to process SOAP 1.1 and SOAP 1.2 messages. You can use the SOAP message handlers in a service requester or in a service provider pipeline.

On input, the SOAP message handlers parse inbound SOAP messages, and extract the SOAP <Body> element for use by your application program. On output, the handlers construct the complete SOAP message, using the <Body> element that your application provides.

If you use SOAP headers in your messages, the SOAP handlers can invoke user-written *header processing programs* that allow you to process the SOAP headers on inbound messages, and to add them to outbound messages.

Note: Do not confuse header processing programs with message handlers. A header processing program can only be invoked by a CICS-supplied SOAP message handler to process a specific kind of SOAP header.

Typically, you will need just one SOAP message handler in a pipeline. However, there are some situations where more than one is needed. For example, you can ensure that SOAP headers are processed in a particular sequence by defining multiple SOAP message handlers.

SOAP message handlers, and any header processing programs, are specified in the pipeline configuration file, using the <cics_soap_1.1_handler> and the <cics_soap_1.2_handler> elements, and their sub-elements.

An example usage of a header processing program is to process a Security header that can be passed in a WS-Security SOAP header. An example of such a header processing program is provided in *Implementing CICS Web Services*, SG24-7206. Appendix B, “A useful header processing program” on page 243 shows another usage of a header processing program, which is to log the contents of the containers passed in a pipeline.

SOAPFAULT commands

SOAP message handlers and SOAP header processing programs can use three API commands that are new in CICS TS V3 to manage SOAP faults:

- EXEC CICS SOAPFAULT CREATE

  Use this command to create a SOAP fault. If a SOAP fault already exists in the context of the SOAP message that is being processed by the message handler, the existing fault is overwritten.

- EXEC CICS SOAPFAULT ADD

  Use this command to add either of the following items to a SOAPFAULT object that was created with an earlier SOAPFAULT CREATE command:

  – A subcode.

  – A fault string for a particular national language.

    If the fault already contains a fault string for the language, then this command replaces the fault string for that language. In SOAP 1.1, only the fault string for the original language is used.

- EXEC CICS SOAPFAULT DELETE

  Use this command to delete a SOAPFAULT object that was created with an earlier SOAPFAULT CREATE command.

These commands require information that is held in containers on the channel of the CICS-supplied SOAP message handler. To use these commands, you must have access to the channel. Only the following types of programs have this access:

- Programs that are invoked directly from a CICS-supplied SOAP message handler, including SOAP header processing programs.
- Programs deployed with the Web services assistant that have a channel interface. Programs with a COMMAREA interface do not have access to the channel.

Many of the options on the SOAPFAULT CREATE and SOAPFAULT ADD commands apply to SOAP 1.1 and SOAP 1.2 faults, although their behavior is slightly different for each level of SOAP. Other options apply to one SOAP level or the other, but not to both, and if you specify any of them when the message uses a different level of SOAP, the command will raise an INVREQ condition. To help you determine which SOAP level applies to the message, container DFHWS-SOAPLEVEL contains a binary fullword with one of the following values:

- 1: The request or response is a SOAP 1.1 message.
- 2: The request or response is a SOAP 1.2 message.
- 10: The request or response is not a SOAP message.

2.6.3 WEBSERVICE

Three objects define the execution environment that allows a CICS application program to operate as a Web service provider or a Web service requester:

- The Web service description
- The Web service binding file
- The pipeline

These three objects are defined to CICS using the following attributes of the WEBSERVICE resource definition:

- WSDLFILE
- WSBIND
- PIPELINE

The WEBSERVICE definition has a fourth attribute, VALIDATION. This attribute specifies whether full validation of SOAP messages against the corresponding schema in the Web service description should be performed at runtime. Validating a SOAP message against a schema incurs considerable processing impact, and you should normally specify VALIDATION(NO) in a production environment. VALIDATION(YES) ensures that all SOAP messages that are sent and received are valid XML with respect to the XML schema. If VALIDATION(NO)

is specified, sufficient validation is performed to ensure that the message contains well-formed XML.

Important: In order to be able to use the VALIDATION option, your CICS region has to be set up with Java™ support. Otherwise, the pipeline processing does not require Java support to be set up for the region.

Figure 2-2 on page 26 and Figure 2-3 on page 30 illustrate the purpose of the WEBSERVICE resource definition for service provider and service requester pipelines respectively.

You may create WEBSERVICE resource definitions in the following ways:

- Using the CEDA transaction
- Using the DFHCSDUP batch utility
- Using CICSPlex SM Business Application Services
- Using the EXEC CICS CREATE WEBSERVICE command

When you install a PIPELINE resource, or when you issue a PERFORM PIPELINE SCAN command (using CEMT or the CICS system programming interface), CICS scans the directory specified in the PIPELINE's WSDIR attribute (the pickup directory), and creates URIMAP and WEBSERVICE resources dynamically. For each Web service binding file in the directory, that is, for each file with the wsbind suffix, CICS installs a WEBSERVICE and a URIMAP if one does not already exist. Existing resources are replaced if the information in the binding file is more recent than the existing resources.

The CEMT INQUIRE WEBSERVICE command is used to obtain information about a WEBSERVICE resource definition. The data returned depends on the type of Web service. Table 2-2 on page 45 shows the types and the data returned for each service.

Table 2-2 CEMT INQUIRE WEBSERVICE command output

Attributes	Service Provider	Service Requester to a local service	Service Requester to a remote service
PIPELINE	Yes	Yes	Yes
VALIDATIONST	Yes	Yes	Yes
STATE	Yes	Yes	Yes
CCSID	Yes	Yes	Yes
URIMAP	Yes, if dynamically installed	Empty	Empty
PROGRAM	Yes	Yes	Empty
PGMINTERFACE	Yes	Yes	No
XOPSUPPORTST	Yes	Yes	Yes
XOPDIRECTST	Yes	Yes	Yes
MAPPINGLEVEL	Yes	Yes	Yes
MINRUNLEVEL	Yes	Yes	Yes
CONTAINER	Yes, if Channel is used	Yes, if Channel is used	No
WSDLFILE	Yes	Yes	Yes
WSBIND	Yes	Yes	Yes
ENDPOINT	Empty	Empty	Yes
BINDING	Yes	Yes	Yes

2.6.4 Web service binding file

Figure 2-7 shows the role of the Web service binding (wsbind) file. The wsbind file is created by the CICS Web services assistant utilities or WebSphere Developer for System z.

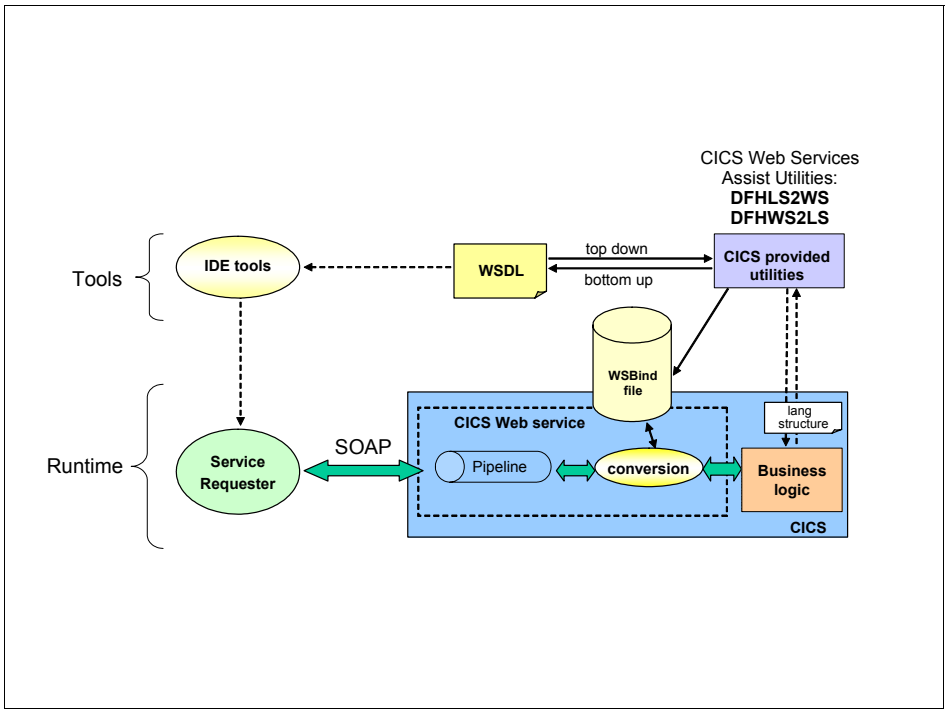


Figure 2-7 wsbind file

The wsbind file is associated with a WEBSERVICE resource and is read when the WEBSERVICE is installed. It contains all of the information necessary to interpret the body of a SOAP message at runtime. This information is used to map data from a SOAP input message to an application data structure (for example, a COMMAREA) and from an application data structure to a SOAP output message.

A sample SOAP message is shown in Example 2-3 on page 47.

Example 2-3 SOAP message

```
Sample SOAP Message

<?xml version='1.0' encoding='UTF-8'?>
<SOAP-ENV:Envelope xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
...>
   <SOAP-ENV:Body>
      <TRGTAPPLOperation>
         <data_structure>
            <structure_part_1>
               <data_item_1>ABCDE</data_item_1>
               <data_item_2>12345678</data_item_2>
               <data_item_3>X</data_item_3>
            </structure_part_1>
            <structure_part_2>
               <data_item_4>Y</data_item_4>
               <data_item_5>ABCDEFG</data_item_5>
               <data_item_6>123</data_item_6>
            </structure_part_2>
         </data_structure>
      </TRGTAPPLOperation>
   </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

The CICS Web services assistant data conversion program converts the XML in Example 2-3 to the data structure shown in Example 2-4.

Example 2-4 Data structure

```
       01  DATA-STRUCTURE.
         02 STRUCTURE-PART-1.
           03  DATA-ITEM-1      PIC X(5).
           03  DATA-ITEM-2      PIC 9(8).
           03  DATA-ITEM-3      PIC X.
         02 STRUCTURE-PART-2.
           03  DATA-ITEM-4      PIC X.
           03  DATA-ITEM-5      PIC X(7).
           03  DATA-ITEM-6      PIC 9(3).
```

The wsbind file is a binary file and contains three sections:

- Header

  The header section contains information about the characteristics of the service, including the target program name for a service provider or the URI for a service requester, and the type of interface (COMMAREA or channel).

- Index

  The index section contains an index of all the operations that are described in this Web service and a pointer to which ICMs describe the data for each operation (input and output).

- Internal COMMAREA Model (ICM)

  An ICM contains the input / output message conversion details for each operation. The purpose of the ICM is to act as an intermediary format for conversion between both (language structures / WSDL) and (SOAP / COMMAREA). It encapsulates sufficient information such that a specific SOAP instance can be converted to a specific COMMAREA or vice versa at runtime.

2.7 Tools for application deployment

In this section, we consider the tools available for application deployment.

2.7.1 CICS Web services assistant

The CICS Web services assistant is a set of batch utilities that can help you to transform existing CICS applications into Web services and to enable CICS applications to use Web services provided by external providers. It contains two utility programs:

- DFHLS2WS

  Generates a Web service binding file from a language structure. This utility also generates a Web service description.

- DFHWS2LS

  Generates a Web service binding file from a Web service description. This utility also generates a language structure that you can use in your application programs.

The assistant supports rapid deployment of CICS applications for use in service providers and service requesters, with a minimum of programming effort. When you use the Web services assistant for CICS, you do not have to write your own code for parsing inbound messages and for constructing outbound messages; CICS maps data between the body of a SOAP message and the application program's data structure.

The assistant can create a WSDL document from a simple language structure, or a language structure from an existing WSDL document. It supports COBOL, C/C++, and PL/I. However, the assistant cannot deal with every possibility, and there are times when you will need to take a different approach. For example:

- You do not want to use SOAP messages.

  If you prefer to use a non-SOAP protocol for your messages, you can do so. However, your application programs will be responsible for parsing inbound messages and constructing outbound messages.

- You want to use SOAP messages, but you do not want CICS to parse them.

  For an inbound message, the assistant maps the SOAP body to an application data structure. In some applications, you may want to parse the SOAP body yourself.

- You have an application written in an unsupported language.

  In this case you should either write a wrapper program in a supported language, or write a program to perform the mapping.

- The CICS Web services assistant does not support your application's data structure.

  Although the CICS Web services assistant supports the most common data types and structures, there are some that are not supported. That is, there may not always be a one to one mapping between XML data types and the data types in your language structure. In this situation, you should first consider providing a "wrapper" program that maps your application's data to a format that the assistant can support. If this is not possible, consider using WebSphere Developer. As a last resort, you may need to change your application's data structure.

If you decide not to use the CICS Web services assistant, you will have to:

- Provide your own code for parsing inbound messages, and constructing outbound messages (unless you use WebSphere Developer)
- Provide your own pipeline configuration file
- Define and install your own URIMAP and PIPELINE resources

MAPPING-LEVEL enhancements

Some restrictions that existed on data types supported by the CICS Web services assistant when CICS TS V3.1 was shipped were removed by APARs PK15904 and PK23547. The MAPPING-LEVEL parameter for both DFHLS2WS and DFHWS2LS specifies the level of mapping that the batch assistant should use when generating the Web service binding file and Web service description. The value of this parameter can be:

1.0 The original default mapping level of CICS TS V3.1.

1.1 APAR PK15904 has been applied to the CICS TS V3.1 region where the Web service binding file is deployed. At this level of mapping, there are improvements to DFHWS2LS when mapping XML character and binary data types, in particular when mapping data of variable length.

1.2 Both APARs PK15904 and PK23547 have been applied to the CICS TS V3.1 region where the Web service binding file is deployed.

2.0 The generated Web service binding file can only be installed into a CICS TS V3.2 region.

At MAPPING-LEVEL1.2, the support for data type mapping that CICS TS V3.1 provides is identical to the one provided by base CICS TS V3.2 at MAPPING-LEVEL 2.0. At this level of support, you can use additional parameters in DFHLS2WS and DFHWS2LS to control how character and binary data is transformed at runtime. You can use the CHAR-VARYING parameter to specify whether to process character fields as fixed length fields or as null terminated strings.

Using DFHLS2WS

This program is typically used when creating a Web service to run in a CICS service provider pipeline, where the COMMAREA or CONTAINER input and output data structures (and perhaps also the program that is to be the target application) already exist. DFHLS2WS takes the input and output data structures from the service provider program, and generates the wsbind and WSDL files.

An example of the JCL required to run DFHLS2WS is shown in Example 2-5 on page 51.

Example 2-5 JCL for DFHLS2WS

```
//LS2WS1   JOB (999,POK),'CICS LS2WS TOOL',MSGCLASS=T,
//             CLASS=A,NOTIFY=&SYSUID,TIME=1440,REGION=0M
//*
//  JCLLIB ORDER=CICSTS32.CICS.SDFHINST
//*
// SET QT=''''
//LS2WS     EXEC DFHLS2WS,
//    JAVADIR='java142s/J1.4',
//    USSDIR='cicsts31',
//    TMPFILE='tmp',
//    PATHPREF=''
//INPUT.SYSUT1 DD *
LOGFILE=/u/cicsdev1/soap1.log
PDSLIB=//CICSDEV1.WEBSERV.SOURCE
REQMEM=DATA-STRUCTURE-INPUT
RESPMEM=DATA-STRUCTURE-OUTPUT
LANG=COBOL
PGMNAME=TRGTAPPL
URI=http://myserver.example.org:8080/example/wsdl/soap11/APPL01
PGMINT=COMMAREA
TRANSACTION=ABCD
WSBIND=/u/cicsdev1/webservices/wsbind/appl01.wsbind
WSDL=/u/cicsdev1/webservices/wsdl/appl01.wsdl
MAPPING-LEVEL=2.0
MINIMUM-RUNTIME-LEVEL=2.0
CHAR-VARYING=NULL
SOAPVER=ALL
WSDL_2_0=/u/cicsdev1/webservices/wsdl/app101_20.wsdl
*/
```

REQMEM and RESPMEM specify the member names containing the input and output data structures, respectively:

- PGMNAME specifies the target application.
- URI specifies the relative or (in this case) an absolute URI that a client will use to invoke this Web service.

  In CICS TS V3.1, you can only specify a relative URI. CICS uses the value specified when it generates a URIMAP resource definition from the wsbind file created by DFHLS2WS. This is the path component of the URI to which the URIMAP definition applies. The complete URI that the Web service HTTP client will use is composed by concatenating the SCHEME (HTTP or HTTPS) followed by HOST and followed by the PATH, as they appear in URIMAP. When the generated WSDL is shipped to the client, it has to be edited to specify the correct host address and port number.

In CICS TS V3.2, you can specify the full URI to the CICS Web Services Assistant, so in this case, there is no need to edit the WSDL file before it is shipped to the client environment.

- PGMINT specifies that the data structure is passed to the application in a COMMAREA. If using a container, specify PGMINT=CONTAINER and add CONTID=container_name as an extra parameter.
- TRANSACTION specifies the name of the alias transaction that can start the pipeline in a service provider or run a user application to compose the response. The value of this parameter is used to define the TRANSACTION attribute of the URIMAP resource when it is created automatically during the PIPELINE scan.
- WSBIND specifies where the wsbind file is to be stored.
- WSDL specifies where the WSDL file is to be stored.
- WSDL_2_0 specifies where the WSDL file that conforms to 2.0 specification will be stored.
- SOAPVER specifies which SOAP level to use in the generated Web service description.
- MINIMUM-RUNTIME-LEVEL specifies the minimum CICS runtime environment on which the Web service bind file can be deployed.
- MAPPING-LEVEL specifies the level of mapping that DFHLS2WS should use when generating the wsbind file and Web service description.
- CHAR-VARYING specifies how character fields in the language structure should be mapped when the mapping level is 1.2 or higher.

Using DFHWS2LS

This program is typically used when creating a Web service to run in a CICS service requester pipeline, where the WSDL to access the service provider already exists. DFHWS2LS takes the WSDL as input, along with the binding to be used (that is, whether to use HTTP or WebSphere MQ at the transport layer), and generates the input and output data structures and the wsbind file to be used by the service requester.

At runtime, the service requester application stores the outbound data structure in a container, issues an EXEC CICS INVOKE WEBSERVICE request to send the Web service request to the service provider, and then reads the inbound data structure response, coming back from the service provider, from the same container.

An example of the JCL required to run DFHWS2LS is shown in Example 2-6.

Example 2-6 JCL for DFHWS2LS

```
//WS2LS1   JOB (999,POK),'CICS WS2LS TOOL',MSGCLASS=T,
//             CLASS=A,NOTIFY=&SYSUID,TIME=1440,REGION=0M
//*
//  JCLLIB ORDER=CICSTS31.CICS.SDFHINST
//*
// SET QT=''''
//WS2LS     EXEC DFHWS2LS,
//    JAVADIR='java142s/J1.4',
//    USSDIR='cicsts31',
//    TMPFILE='tmp',
//    PATHPREF=''
//INPUT.SYSUT1 DD *
LOGFILE=/u/cicsdev1/soap1.log
PDSLIB=//CICSDEV1.WEBSERV.SOURCE
REQMEM=DATA-STRUCTURE-INPUT
RESPMEM=DATA-STRUCTURE-OUTPUT
LANG=COBOL
BINDING=APPL01HTTPSoapBinding
WSBIND=/u/cicsdev1/webservices/wsbind/appl01.wsbind
WSDL=/u/cicsdev1/webservices/wsdl/appl01.wsdl
TRANSACTION=XYZW
MAPPING-LEVEL=2.0
MINIMUM-RUNTIME-LEVEL=2.0

*/
```

These are the main differences in the parameters between DFHLS2WS and DFHWS2LS:

- REQMEM and RESPMEM specify the member names for the data structures to be created, while these were input parameters for DFHLS2WS.
- PGMNAME and PGMINT are omitted.
- URI is omitted. The URI to invoke this Web service is in the WSDL.
- WSDL specifies where the WSDL file is located.
- TRANSACTION specifies the name of the alias transaction that can start the pipeline in a service provider, or run a user application to compose the response. The value of this parameter is used to define the TRANSACTION attribute of the URIMAP resource when it is created automatically during the PIPELINE scan.
- BINDING specifies which binding name within the WSDL is to be used. This is how either HTTP or WebSphere MQ is selected by a CICS client.

- MINIMUM-RUNTIME-LEVEL specifies the minimum CICS runtime environment on which the Web service bind file can be deployed.
- MAPPING-LEVEL specifies the level of mapping that DFHLS2WS should use when generating the wsbind file and Web service description.

2.7.2 WebSphere Developer for System z

IBM WebSphere Developer for System z V7 combines the power of service-oriented architectures that use Java 2 Enterprise Edition (J2EE™), CICS, IMS, COBOL, and PL/I technologies with a Rapid Application Development (RAD) paradigm and teaming, and it brings this power to diverse enterprise application development organizations.

Today's applications are not developed by super technologists in a vacuum. Instead, they are developed by teams of people with varying levels of technology backgrounds but with the following common goals:

- A firm understanding of the business
- A requirement to integrate across many business processes
- A desire to leverage currently executing (as well as new) Web-oriented technologies
- A desire to promote usage of the Internet in meeting business needs
- A desire to leverage development standards and expertise to move the business forward

Two overriding goals drive these organizations:

- Meeting requirements of time to market
- Meeting high quality standards

To help enterprise customers meet these goals, IBM delivered WebSphere Developer for System z. WebSphere Developer for System z supports a development process that includes:

- A standards-based Java server faces visual environment that supports rapid application development (RAD).
- A visual Web-flow construction environment built on the Struts-based Model-View-Controller (MVC) paradigm. The elements of the MVC paradigm are as follows:
  - Model and Business Logic: Invoked by or implemented in Struts action classes. Business logic can be implemented in a variety of technologies, such as Java, COBOL, and PL/I.

  - View: Implemented using HTML and Java Server Pages with special Struts tags.
  - Controller: Flow of Web applications, implemented as Struts actions or actions defined as supporting various underlying business and technical processes.

  The MVC paradigm is recommended by J2EE experts, and used by architects and developers to define, reuse, and debug application organization, flow, and actions.

- Editors and interactive development environments (IDEs) for Java, COBOL, PL/I, and Enterprise Generation Language (EGL) components, including language understanding, syntax checking, and unit testing in WebSphere, CICS, and IMS transactional environments.
- Web services support, including client generation, XML editors, COBOL and PL/I adapter generation, and support for Web services deployed to WebSphere, CICS, and IMS, through SOAP for IMS.
- Support for multiple databases, including DB2®.
- Integration of Web services and XML processes, including transformation of messages in WebSphere, CICS, and IMS transactional environments.
- COBOL transformers.
  - Generated from XML Schema Definition (XSD) mapping leveraging IBM Enterprise COBOL parsing and generation.
  - Deployable to CICS, IMS, and batch runtime environments.
- Support for Java and COBOL stored procedure generation and deployment.
- Remote z/OS support, including data set access, job submission, queue management, and TSO command processing.
- Cross platform interactive testing and debugging, including WebSphere, CICS, and IMS transactional environments.
- Deployment in WebSphere, CICS, and IMS transactional environments.
- Local CICS development environment.

In short, the goal of WebSphere Developer is to lower the technological requirements of entry to modern application development, and to embrace a service-oriented architecture (SOA), leveraging a reusable model-based development paradigm that helps organizations meet their business needs today and in the future. Organizations that use WebSphere Developer should be able to:

- Simplify the process of developing J2EE and integrated mixed workload applications that include CICS and IMS processing.

- Include and support the collaboration of multiple roles in the process.
- Promote reuse and transformation of existing applications to an e-business model.
- Lessen training costs and leverage common developer skills across mixed workload environments, thus increasing overall organizational knowledge and flexibility.

2.7.3 Comparing Web services assistant and WebSphere Developer for System z

Table 2-3 compares the XML parsing and generation capabilities of the CICS Web services assistant at mapping level 1.2 or 2.0 with those of the WebSphere Developer for System z V7.

Table 2-3 CICS Web services assistant and WebSphere Developer parsing/generation

Description	**Web services assistant at mapping levels 1.2 or 2.0**	**WebSphere Developer for System z V7**
Implementation	ICM - metadata	Generated Cobol parser
Language support	► COBOL ► PL/I ► C or C++	COBOL
COBOL data types	► DISPLAY ► Alphanumeric edited ► Numeric edited ► COMP ► COMP-1 ► COMP-2 ► COMP-3 ► COMP-4 ► COMP-5 ► BINARY	► DISPLAY ► Alphanumeric edited ► Numeric edited ► COMP ► COMP-1 ► COMP-2 ► COMP-3 ► COMP-4 ► COMP-5 ► BINARY
COBOL structure	Elementary items	► Elementary items ► 88 level items ► Group items ► Redefines ► OCCURS ► OCCURS DEPENDING ON

Description	Web services assistant at mapping levels 1.2 or 2.0	WebSphere Developer for System z V7
C and C++ data types	► char ► unsigned char ► signed char ► short ► unsigned short ► int ► unsigned int ► long ► unsigned long ► long long ► unsigned long long ► bool (C++ only) ► float ► double	N/A
PL/I data types	► FIXED BINARY ► UNSIGNED FIXED BINARY ► FIXED DECIMAL ► BIT ► CHARACTER ► GRAPHIC ► WIDECHAR ► ORDINAL ► BINARY FLOAT ► DECIMAL FLOAT	N/A

3

CICSPlex SM overview

In this chapter, we review the concepts of CICSPlex SM, and we discuss the basic components of the CICSPlex SM Version 3 Release 2, before taking a look at the CICSPlex SM Web User Interface in detail.

3.1 CICSPlex SM introduction

The CICSPlex System Manager element of CICS Transaction Server for z/OS Version 3 Release 2 (CICSPlex SM) is a system management tool that enables you to manage multiple CICS systems across multiple images from a single control point. Enterprises in which CICSPlex SM may be needed range from those running only a few CICS systems, to those running several hundred (or more) CICS systems. In the latest MVS™ sysplex environment, having such large numbers of CICS systems to support a transaction-processing workload is becoming an increasing requirement.

Each CICS region to be managed by CICSPlex SM is called a managed application system (MAS). The MASs are defined and managed as part of a CICSplex. Each MAS in a CICSplex is managed by a CICSPlex SM address space (CMAS).

To run the CICSPlex SM component of CICS Transaction Server for z/OS Version 3 Release 2, a MAS may be executing CICS Transaction Server for z/OS Version 2 Release 2, Version 2 Release 3, Version 3 Release 1, or Version 3 Release 2 on a system running an MVS image. However, the CMAS and the WUI server must execute the same release of CICS TS and CICSPlex SM.

The MASes in a CICSplex can be managed by several CMASs, but only one CMAS is defined as the maintenance point (MP) CMAS. This MP CMAS is responsible for keeping the data used by each CMAS synchronized.

CMASs communicate across defined CMAS-to-CMAS links, which are typically used for routing management commands and data between CMASs. The CICSPlex SM Web User Interface (WUI) is now, since CICS Transaction Server for z/OS Version 3 Release 2, the only user interface provided (the MVS/TSO ISPF user interface is no longer provided). The WUI is a server that runs on a dedicated CICSPlex SM local MAS at the same CICS Transaction Server release level as the connected CMAS.

Resource definitions are managed through Business Application Services (BAS). Workload management (WLM), Real Time Analysis (RTA), and monitoring services are used to manage the CICSPlex SM configuration and CICSplex environment, and gather statistical information.

All CICSPlex SM components, resources, system management requirements, and the relationships between them are held as objects in a data repository. These objects can be manipulated using the WUI user interface views. The batched repository-update facility is provided for the batched creation of CICSPlex SM resource definitions. Figure 3-1 on page 61 shows an overview of

the components in the CICSPlex SM. These are all discussed in 3.2, “Basic CICSPlex SM components” on page 62.

For CICSPlex SM’s purposes, a CICSplex is any grouping of CICS systems that you want to manage and manipulate as though they were a single entity. That is, a CICSplex is a management domain, made up of those CICS systems for which you want to establish a single system image (SSI). A CICSplex managed by CICSPlex SM could include every CICS system in your enterprise, or alternatively, you could define multiple CICSplexes. Each of these would include a logical grouping of CICS systems. For example, a CICSplex could comprise all CICS systems on a particular MVS image, or all CICS systems accessible by a subset of your users. It could perhaps even be all CICS systems serving a particular geographical area. Furthermore, the composition of a CICSplex can be altered without affecting the functions of the underlying CICS systems. The CICS systems in a single CICSplex managed by CICSPlex SM do not have to be explicitly connected to each other for management purposes.

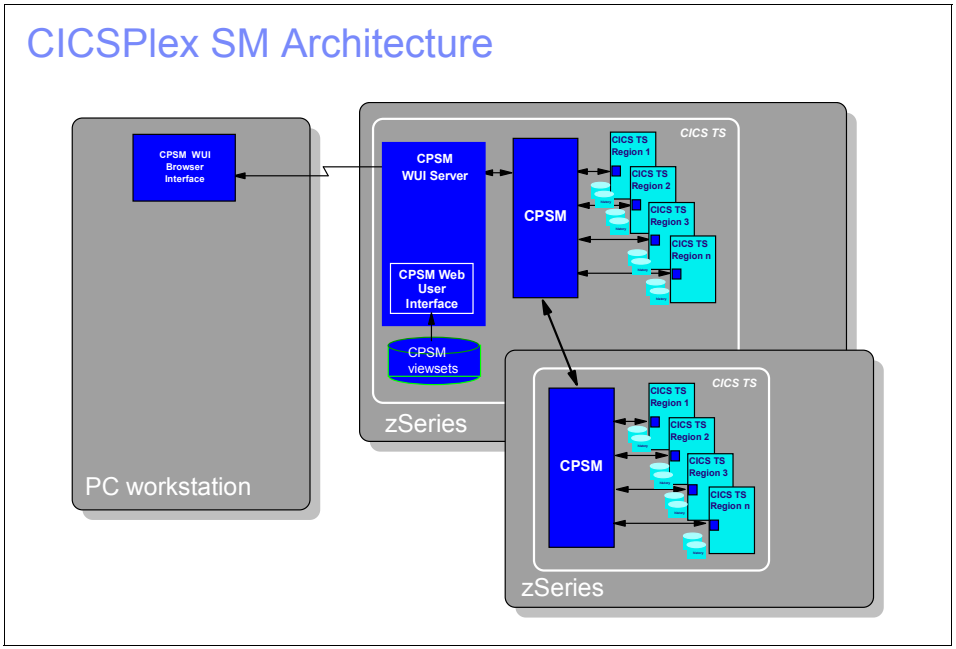


Figure 3-1 CICSPlex SM components

The most significant facts about a CICSplex managed by CICSPlex SM are:

- A CICSplex could be just on a single MVS image, or on a sysplex or multiple sysplexes (for example, spanning multiple countries).
- The CICSplex is the largest unit that can be managed from a single point of control. That is, you cannot group CICSplexes and manipulate such a group as a single entity.
- You cannot copy CICSPlex SM data from one CICSplex to another. For system management purposes, the CICSplex is “sealed” against other CICSplexes.
- A CICSPlex SM managed CICS system can only be active in one CICSplex at a time.

CICSPlex SM enables you to define subsets of a CICSplex, which are known as CICS system groups. CICS system groups are not mutually exclusive, and can reference the same CICSPlex SM definitions. Thus, if you decide to include every CICS system in your enterprise in a single CICSplex, there are mechanisms for managing groups of CICS systems within the CICSplex as though each group were a single system. You can assign an unlimited number of CICS systems and CICS system groups to an existing CICSplex. Although you can define a CICS system to only one CICSplex, you can assign a CICS system to multiple CICS system groups within the CICSplex. You can also assign the CICS system group to any number of other CICS system groups.

Important: CICS TS V3.2 requires z/OS (5694-A01) V1.7 or later.

3.2 Basic CICSPlex SM components

Running CICSPlex SM for CICS Transaction Server for z/OS Version 3 Release 2 requires, at minimum, three dedicated address spaces:

- CICSPlex SM Address Space (CMAS)
- CICSPlex SM Web User Interface (WUI)
- Environment Services System Services (ESSS)

The CMAS and the WUI server are both dedicated CICS regions. They do not run or contain application code, and should be used solely for CICSPlex SM.

A CICS region that becomes part of a CICSplex is known as a Managed Application System (MAS).

Note: We recommend that in a real production environment the WUI server be placed in a CICS region as a dedicated MAS for CICSPlex SM WUI services.

CMAS overview

The CMAS is a CICS region dedicated solely to the CICSPlex SM function. It is responsible for the managing and reporting of all CICS regions and resources within the defined CICSplex(es).The CMAS interacts with CICSPlex SM agent code running on each managed CICS region (MAS) to define events, conditions of interest, collect information gathered or to report on as a result of such definitions.

A maintenance point (MP) CMAS is the owner of a CICSplex. When more than one CMAS is involved in managing a CICSplex, then the CMAS that was used to create and define the CICSplex becomes the maintenance point. This is basically the master repository for the CICSplex. For the permanent modification to the environment or repository, the maintenance point CMAS must be available.

Important: The maintenance point for a CICSplex cannot be changed without deleting and redefining the whole plex. Make sure you know that the CMAS within which you are defining the plex is on the correct one.

The CICSplex topology (BAS, MON, WLM, and RTA definitions) and configurational definitions are all stored in the CMAS repository data set, EYUDREP.

In more complex environments, multiple CMAS regions can communicate between MVS regions to make up CICSplexes. These may consist of CICS regions across a sysplex or located at geographically separate sites.

For simplicity, we only consider a single CMAS for the Web User Interface server and all MAS regions here.

Note: A CMAS region is not part of a managed CICSplex although it manages one or more CICSplexes.

MAS overview

To be registered as a MAS to CICSPlex SM, a CICS region requires the following to be defined in the startup procedure:

- CICSPlex SM agent load libraries (SEUYLOAD and SEYUAUTH)
- CICSPlex SM parameter defining to which CMAS the CICS will connect to (a DD statement called EYUPARM in the CICS startup procedure)
- DFHSIT parameter denoting the type of CICS connection to be made (CPSMCONN=LMAS)

Note: DFHPLT program EYU9XLCS is now no longer required in the MAS.

The CICSPlex SM agent code is executed during CICS initialization to register the MAS with its respective managing CMAS.

Agent code running on the MAS communicates pertinent statistical and monitoring data back to the CMAS to which it connects.

Note: MASes do not have to run the latest version of CICS code.

The CICSPlex SM Web User Interface server

Traditionally, CICSPlex SM user interaction took place through the MVS/TSO ISPF user interface (EUI). However, the Web browser-based interface is now the *only* user interface to CICSPlex SM. This requires the use of a second dedicated CICS region known as a Web User Interface (WUI) region. The Web User Interface makes use of CICS Web Services to allow interaction with a Web Browser over TCP/IP. For more information about this topic, see 3.3, “CICSPlex SM Web User Interface” on page 64.

A page of information from the Web User Interface region presented on the browser is known as a *WUI view*; a collection of Web User Interface views is known as a *View set*.

The ESSS

There is one further address space that is automatically created upon startup of the CMAS, called the Environment Services System Services (ESSS). It is a limited function system address space that provides MVS system services to the CICSPlex SM components.

Note: There is one ESSS per CICSPlex SM release per MVS image, that is, you will have multiple ESSSs when migrating to a new release.

3.3 CICSPlex SM Web User Interface

The CICSPlex SM Web User Interface (WUI) offers an easy-to-use interface that you can use to carry out operational and administrative tasks necessary to monitor and control CICS resources. You can link to the Web User Interface from any location that provides IP network connectivity and firewall security access to a host from a workstation to the WUI server.

3.3.1 CICSPlex SM Web User Interface overview

The WUI is supplied with a set of linked menus and views to facilitate all your system management tasks.

The WUI can also be customized to reflect your business procedures or to suit the needs of individual users.

The CICSPlex SM Web User Interface allows you to:

- Create clear, uncluttered menus and displays (called views) that present only the information that you wish the user to see.
- Structure your data in a task-oriented way. You can:
  - Organize the user interface by resource category, by user task, or by application.
  - Define the links between views.
  - Define the buttons that will appear on a display and what they will do.
- Customize the layout of data. You can:
  - Have as many views of the same object as you like, each one showing a different selection of data depending on the user task.
  - If you have a Java-enabled browser, you can use graphical presentations of your data: You can have either a bar gauge that shows, for example, the number of tasks active in a CICS region, or a warning light that can be configured to change color or flash, depending on the threshold values you define for the field.
- Customize the panels to your business needs. You can:
  - Use terminology appropriate to your business.
  - Limit the data that is displayed using filters, so that users see only the data relevant to their task.
  - Include information for the user's guidance, for example, contact names and telephone numbers.
  - Define text that is written on action buttons.
  - For each menu choice, add explanatory text to help the user in the task.
  - For each view, provide buttons that accomplish a specific task, for example, a shutdown button on a CICS regions view.
- Assign views to a set of favorites for quick and easy access. This allows you to reach frequently used views with just one click. Administrators have the additional authority to update and maintain the favorites of other users.

- Present the data the users want to see in order to complete a task. You can:
  - Create profiles for groups of users. These profiles contain information such as default context, scope, CMAS context, menu, and result set warning count. In this way administrators can configure the WUI in different ways to suit different groups of users in order to present an interface that is more tailored to individual needs.
  - Display only the information you want the user to see.
  - Control what information can be amended, and where and how these amendments are made. For example, you can make sure that the user has to confirm that an operation is required, or that data has to be changed. You can restrict entry fields to display-only or to preset values.
  - Add safety by providing a confirmation panel asking the user to confirm that an action is to be performed.
  - Set the WUI to issue warnings before it opens a view that will generate a large numbers of records. This improves performance by reducing unnecessary waits.
- Develop menus that guide the user through a task. For each of the tasks being performed in your enterprise, you know which CICSPlex SM objects are involved in the task, and so you can create a menu for the task that contains those objects. In this way, you can create menus that reflect your business procedures.
- Protect the view editor, user editor, and specific menus, views, and help panels from unauthorized access, thus protecting the environment.

Note: Browsers that support HTML V4:

- Microsoft Internet Explorer® 6.0 and 7.0
- Mozilla Firefox 2.0.

Further reading

More complex environments may require further background reading, such as *CICSPlex SM for z/OS Concepts and Planning,* SC34-6015.

3.3.2 CICS TS V3.2 WUI enhancements

In this section, we discuss CICS TS V3.2 WUI enhancements.

Help Information

Enhanced, consistent information, with reduced button length and column headings, enabling improved window layout. Three levels of help are provided:

- General WUI help
- The CICS Information Center (available if INFOCENTER is coded in the SIT parameter for WUI)
- View specific help

Summary Views

These are improved to enable the display of details of summarized records through a new link.

Map Support

This is now added as a “button” facility allowing the user to explore the associations between administrative resource definitions in an interactive diagrammatic manner. This facility replaces the ISPF Interface “MAP” function.

Export Facility

The COVC transaction that is used for the management of WUI now enables the exporting of the entire WUI repository by means of the addition of the “ALL” parameter. In addition, new server messages have been introduced for this function.

3.4 CICSPlex SM installation enhancements

Note: The EYUDREP repository defined in the CMAS (not WUI) still remains the primary central repository in the CICSplex.

The CICSPlex SM installation has been simplified by adding the process into the one used for CICS. Thus, instead of having to run EYUISTAR as before (which is now no longer available), the CICSPLex SM installation process is now included in DFHISTAR during the CICS upgrade or installation process.

Previously, all definitions that were required to be added manually in the CSD (perhaps by the “UPGRADE USING EYU96xG0” parameter of the utility DFHCSDUP) are now installed dynamically both at initialization time, and when required during runtime (by the program EYU9XLCD). This applies to the CMAS, WUI, and the CICSplex agents in the MAS regions managed by CICSPlex SM.

The EYU9XDUT utility can now also be used to create definitions required to start a Web User Interface and its associated CICSplex for the first time (having never been installed before), replacing the TSO user interface.

CICSPlex SM FMID has been changed to be dependant on the CICS FMID. This allows mutual prerequisites between CICS and CICSPlex SM for PTF management.

Note: We advise defining the MP for all managed CICSplexes as a stand-alone CMAS that does not have any MASes connecting to it. This eases application of maintenance to CICSPlex SM for future upgrades.

3.5 CICSPlex SM enhancements

The following are functional enhancements introduced to CICSPlex SM V3.2:

- The new EYU9XDBT utility simplifies the setup (an easy-to-use API command interface) and can be used as an alternative to the BATCHREP facility. It can be used once the basic CMAS environment has already been established. Some examples are provided in SEUYSAMP:
  - EYUJXBT0: JCL syntax for quick reference
  - EYUJXBT1: Sample JCL to define CICSPlex, CICs system group, and a CICS system definition
  - EYUJXBT2: Sample to create a CMAS to CMAS link definition
- Enhancements to WUI views.
- Support for dynamic program library management through new LIBRARY, LIBDSN, and LIBDEF resource tables.
- Full functional support for IP interconnectivity for DPL with the new IPCONN and IPCONDEF resource tables.
- Enhancements to TDQ view sets in WUI.
- “CMAS in CICSPlex definitions view” is enhanced to display all the CMASs that manage CICSPLEXs for which the context is the MP.
- Improved performance class monitoring information can now be obtained.

- Consideration has now been made within the workload algorithm to accommodate workload balancing for dynamic routing over IP connections.
- New attributes have been added to existing views, and the related resource tables have been updated for 64-bit support.
- New views and resource tables have been added to accommodate 64-bit support.

3.6 CICSPlex SM WUI and CICS Web services

In this section, we will look at the WUI views that are available to help install and manage your CICS Web services applications.

The following WUI views are available with CICS TS V3:

- URI map - URIMAP
- Global URI map statistics - URIMPGBL
- URI host - HOST
- Web service - WEBSERV
- Pipeline - PIPELINE

To see all of the CICS Web services operations views, go the CICS operations views in the TCP/IP service operations views on your WUI, as shown in Figure 3-2.

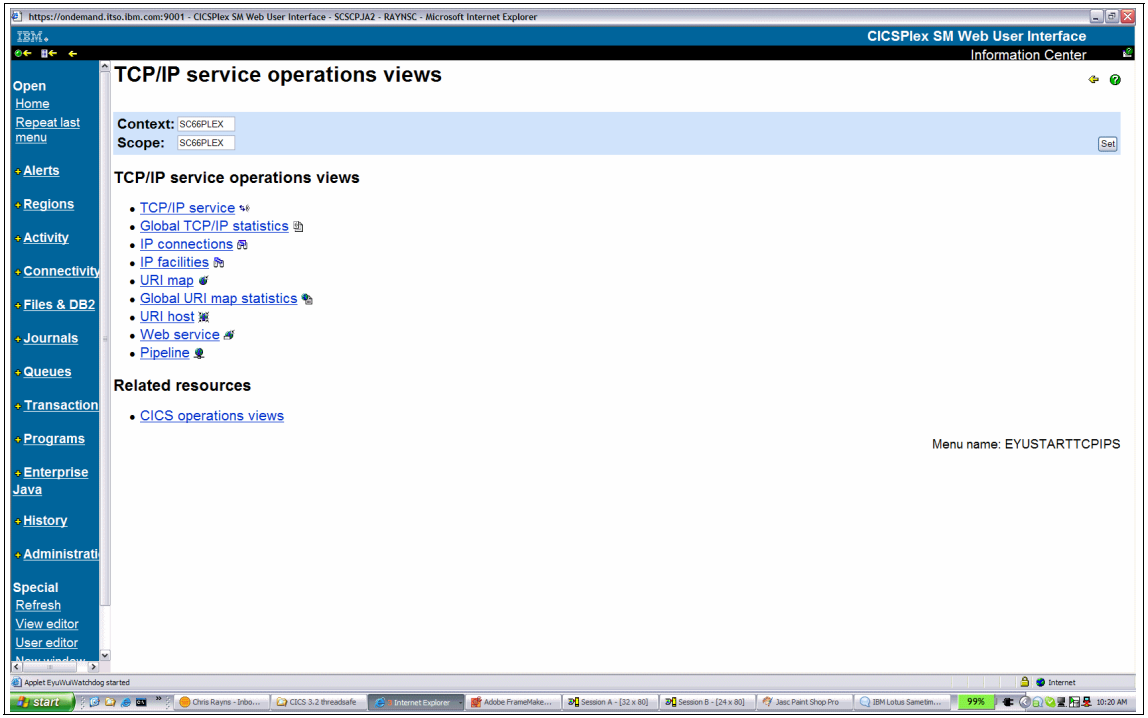


Figure 3-2 TCPI/IP service operations views

3.6.1 URI map - URIMAP

The URI map (URIMAP) views display information about the universal resource identifier (URI) of requests from Web clients or requests to a remote server. To get to the URIMAP window, click **URImap** in the TCP/IP service operations view. See Figure 3-3 on page 72 for an example window that shows URImap.

Supplied views

Table 3-1 on page 71 shows the views in the supplied URI map view set.

Table 3-1 Views in the supplied URI map (URIMAP) view set

View	Notes
URI map EYUSTARTURIMAP.DISABLE	Disable program access to the URIMAP definition. A URIMAP definition has to be disabled before it can be reinstalled or discarded.
URI map EYUSTARTURIMAP.DISCARD	Remove a URIMAP definition from the system.
URI map EYUSTARTURIMAP.TABULAR	Tabular information about currently installed URI map definitions.
URI map EYUSTARTURIMAP.DETAIL2	Detailed information about a selected URI map.
URI map EYUSTARTURIMAP.DETAILED	Detailed information about a selected URI map.
URI map EYUSTARTURIMAP.ENABLE	Enable access to the URIMAP definition by programs.
URI map EYUSTARTURIMAP.SET	Set attributes according to the new values specified in the input fields.
URI map EYUSTARTURIMAP.DETAIL1	Detailed information about a selected URI map.

Action

Table 3-2 shows the actions available for the URIMAP views.

Table 3-2 Actions available for URIMAP views

Action	Description
Disable	Disable program access to the URIMAP definition. A URIMAP definition has to be disabled before it can be reinstalled or discarded.
Discard	Remove a URIMAP definition from the system.
Enable	Enable access to the URIMAP definition by programs.
Set	Set attributes according to the new values specified in the input fields.

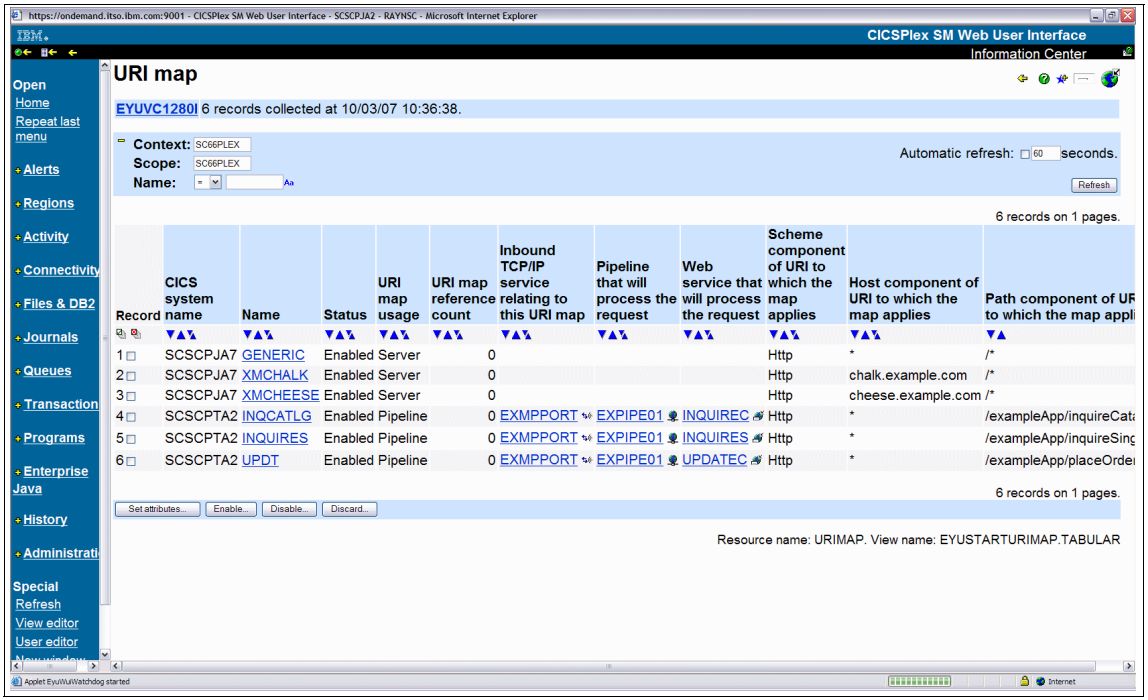


Figure 3-3 URImap

3.6.2 Global URI map statistics - URIMPGBL

The Global URI map statistics (URIMPGBL) views display the global statistics returned by CICS extract statistics for URIMAP resources, as shown in Figure 3-4 on page 73.

Supplied views

Table 3-3 shows the views in the supplied Global URI map statistics (URIMPGBL) view set.

Table 3-3 Views in the supplied Global URI map statistics (URIMPGBL) view set

View	Notes
Global URI map statistics EYUSTARTURIMPGBL.TABULAR	Tabular information about URI map global statistics for all CICS systems
Global URI map statistics EYUSTARTURIMPGBL.DETAILED	Detailed information about URI map global statistics for a selected CICS system

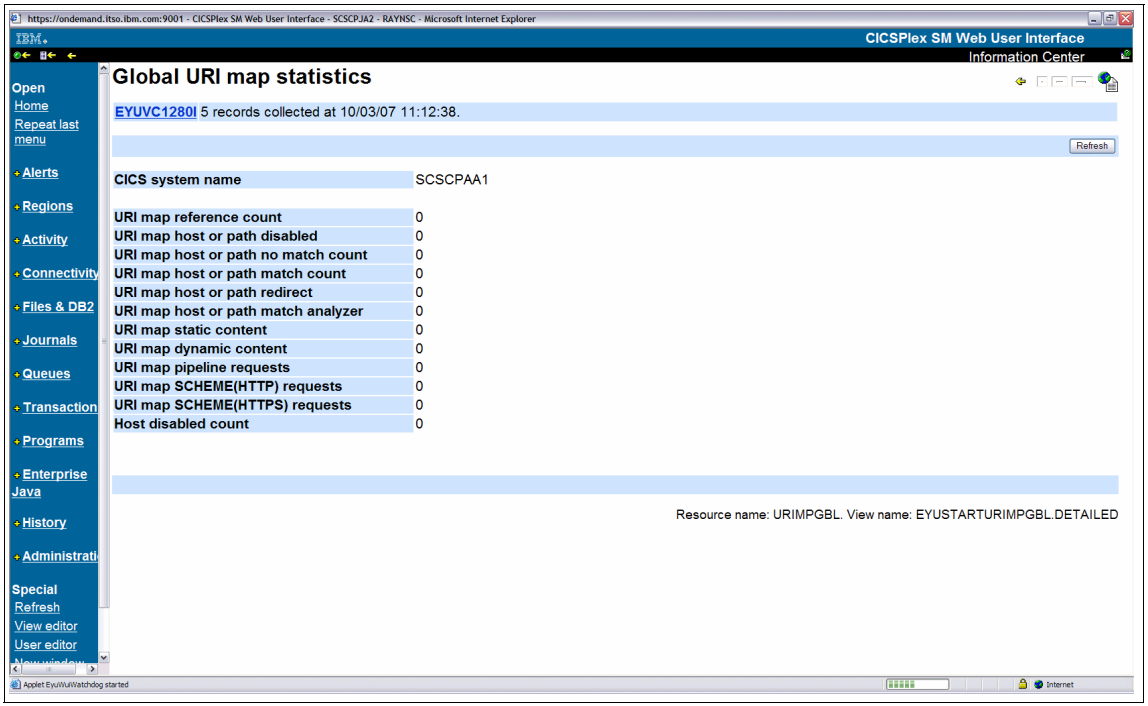


Figure 3-4 Global URI map

3.6.3 URI host - HOST

The URI map (URIMAP) views display information about the universal resource identifier (URI) of requests from Web clients or requests to a remote server, as shown in Figure 3-5 on page 74.

Supplied views

Table 3-4 shows the views in the supplied URI host (HOST) view set.

Table 3-4 Views in the supplied URI host (HOST) view set

View	Notes
URI host EYUSTARTHOST.DISABLE	Disables a host.
URI host EYUSTARTHOST.TABULAR	Tabular information about virtual hosts in the local system.
URI host EYUSTARTHOST.DETAILED	Detailed information about a selected virtual host.

View	Notes
URI host EYUSTARTHOST.ENABLE	Enables a host.
URI host EYUSTARTHOST.SET	Sets attributes according to the new values specified in the input fields.

Actions

Table 3-5 shows the actions available for HOST views.

Table 3-5 Actions available for HOST views

Action	Description
Disable	Disables a host.
Enable	Enables a host.
Set	Sets attributes according to the new values specified in the input fields.

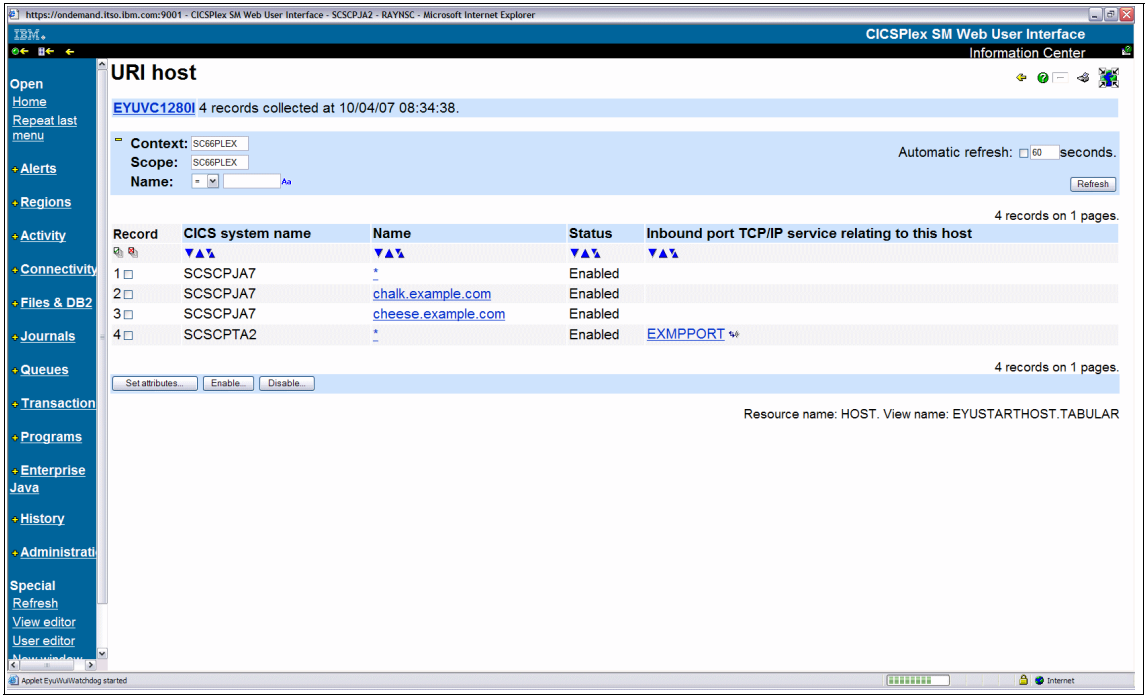


Figure 3-5 URI host

3.6.4 Web service - WEBSERV

The WEBSERV views display information about the runtime environment for a CICS application programs deployed in a Web services setting, where the mapping between application data structure and SOAP messages has been generated using CICS-supplied tools.

Supplied views

Table 3-6 shows the views in the supplied Web service (WEBSERV) view set.

Table 3-6 Views in the supplied Web service (WEBSERV) view set

View	Notes
Web service EYUSTARTWEBSERV.DISCARD	Discards a Web service.
Web service EYUSTARTWEBSERV.TABULAR	Tabular information about Web services.
Web service EYUSTARTWEBSERV.DETAILED	Detailed information about a selected Web service.
Web service EYUSTARTWEBSERV.SET	Sets attributes according to the new values specified in the input fields.

Actions

Table 3-7 shows the actions available for WEBSERV views.

Table 3-7 Actions available for WEBSERV views

Action	Description
Discard	Discards a Web service
Set	Sets attributes according to the new values specified in the input fields.

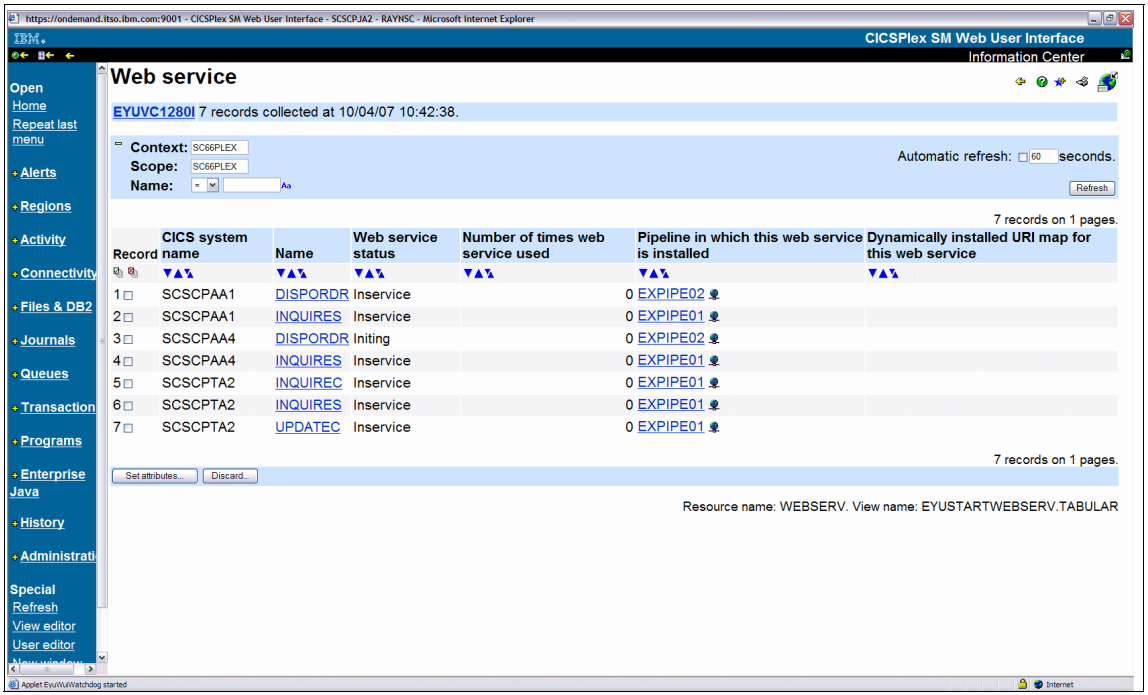


Figure 3-6 Web service view

3.6.5 Pipeline - PIPELINE

The Pipeline (PIPELINE) views display information about the processing nodes that will act on a service request and on the response to it when a CICS application acts in the role of a Web service provider or requester. See Figure 3-7 on page 78 for a screen capture of the pipeline view from the WUI.

Supplied views

Table 3-8 shows the views in the supplied Pipeline (PIPELINE) view set.

Table 3-8 Views in the supplied Pipeline (PIPELINE) view set

View	Notes
Pipeline EYUSTARTPIPELINE.DISABLE	Rejects inbound service requests.
Pipeline EYUSTARTPIPELINE.DISCARD	Removes this PIPELINE definition.

View	Notes
Pipeline EYUSTARTPIPELINE.TABULAR	Tabular information about the pipeline.
Pipeline EYUSTARTPIPELINE.DETAILED	Detailed information about a selected pipeline.
Pipeline EYUSTARTPIPELINE.ENABLE	Processes inbound service requests normally.
Pipeline EYUSTARTPIPELINE.SET	Sets attributes according to the values specified in the input fields.
Pipeline EYUSTARTPIPELINE.SCAN	Scans the PIPELINE's Web service binding directory.

Actions

Table 3-9 shows the actions available for PIPELINE views.

Table 3-9 Actions available for PIPELINE views

Action	Description
Disable	Rejects inbound service requests.
Discard	Removes this PIPELINE definition.
Enable	Processes inbound service requests normally.
Set	Sets attributes according to the values specified in the input fields.
Scan	Scans the PIPELINE's Web service binding directory.

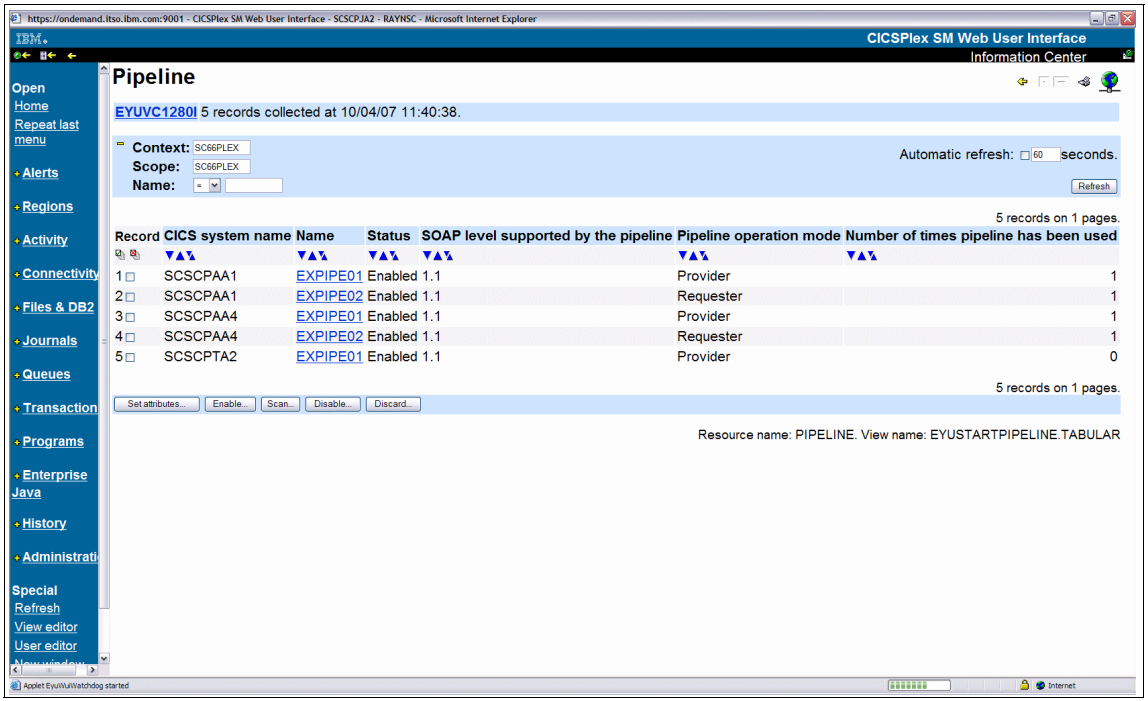


Figure 3-7 Pipeline view

4

Workload management and availability

In this chapter, we discuss the different techniques that can be used to provide high system availability and workload management for CICS Web service applications.

We discuss how high availability is provided across a Parallel Sysplex where TCP/IP traffic uses the following functions:

- Port Sharing
- Virtual IP Addressing
- Sysplex Distributor

Next, we describe how CICSPlex SM can be used to manage a Web services workload using dynamic program link or transaction routing. We also describe the technique for routing Web service requests to regions dedicated to invoking Web services.

4.1 Workload balancing

As the service hit rate increases or because of availability demands, it may become necessary to balance a Web services workload across multiple CICS regions. For HTTP, this can be achieved by using port sharing or the Sysplex Distributor to route the incoming requests to different CICS regions within a Sysplex. For WebSphere MQ, a queue sharing group (described in *WebSphere MQ for z/OS Concepts and Planning Guide V5.3.1,* GC34-6051) can be set up to allow multiple CICS regions to service messages on the same queue.

Once within CICS, the existing business logic application that is linked from the message adapter can be on an application owning region (AOR) and workload managed, for example, by CICSPlex SM.

Figure 4-1 shows an example of workload balancing across multiple regions.

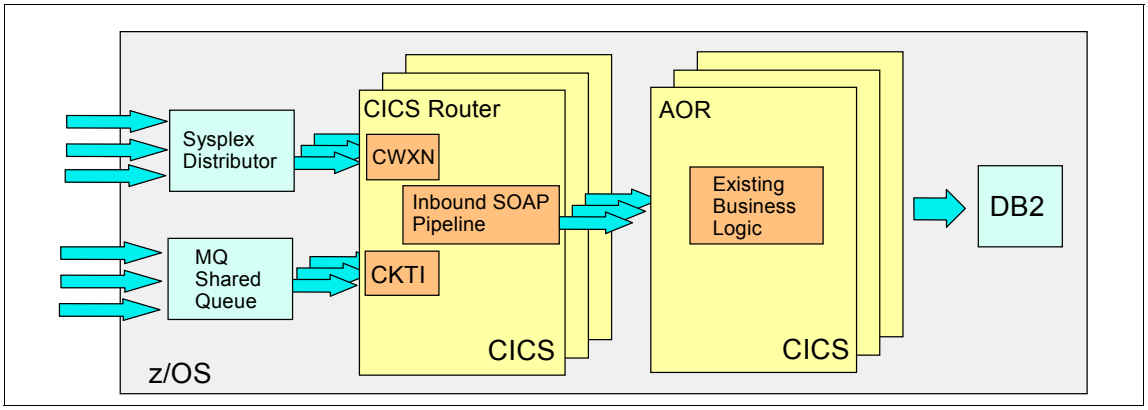


Figure 4-1 Workload balancing across multiple regions

Note: In this book, we only discuss high availability configurations for Web services using HTTP.

4.2 TCP/IP load balancing techniques

In this section, we discuss TCP/IP load balancing techniques, looking at port sharing, virtual IP addressing, and the Sysplex Distributor.

4.2.1 Port sharing

TCP/IP port sharing provides a simple way of spreading HTTP requests over a group of CICS router regions running in the same z/OS image. CICS TCPIPSERVICEs in different regions are configured to listen on the same port, and TCP/IP is configured with the SHAREPORT or SHAREPORTWLM options. The TCP/IP stack then balances connection requests across the CICS router regions.

The example shown in Figure 4-2 illustrates the use of TCP/IP port sharing to achieve workload balancing between two front-end CICS regions (PTA1 and PTA2) running on the same LPAR.

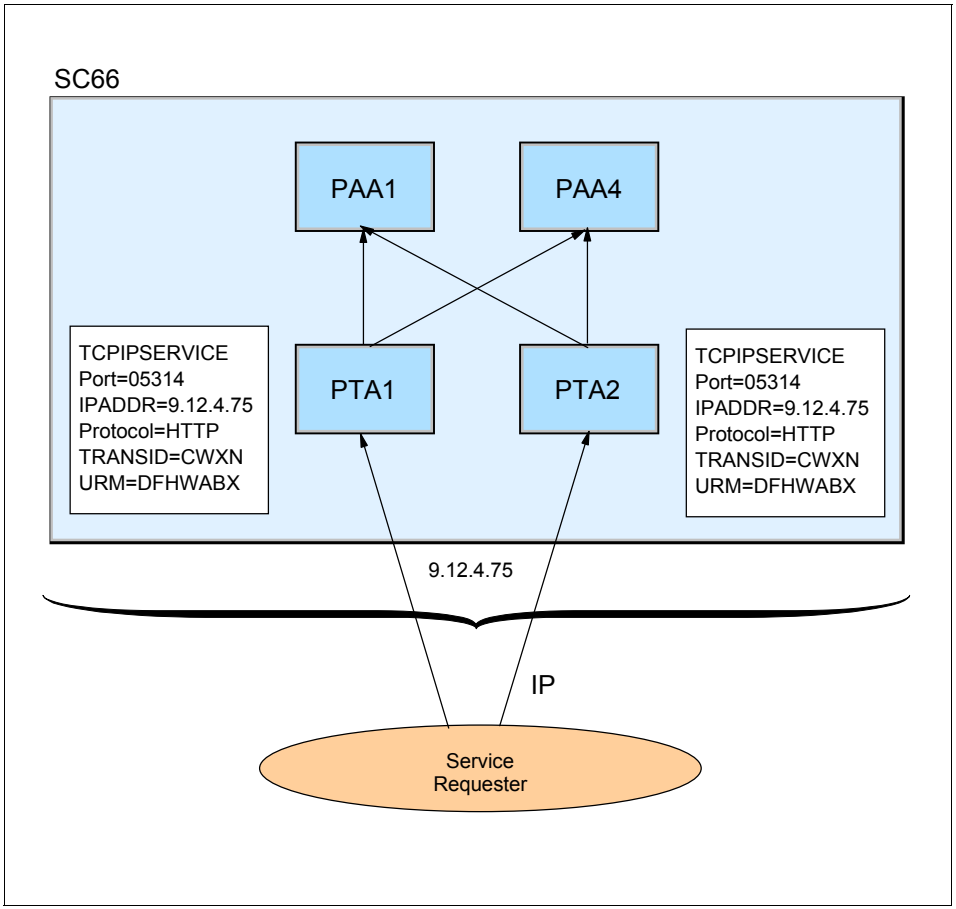


Figure 4-2 TCP/IP port sharing example

When SHAREPORT is specified on the PORT statement in the TCP/IP profile, TCP/IP evenly balances the number of active connections across the available servers based on the number of active and backlog socket connections.

The SHAREPORTWLM option can be used instead of SHAREPORT. Like SHAREPORT, SHAREPORTWLM causes incoming connections to be distributed among a set of servers; however, unlike SHAREPORT, the listener selection is based on WLM server-specific recommendations.

Using port sharing spreads the SOAP request messages across multiple CICS regions and therefore improves availability. There remains, however, a single point of failure in the event of an IP stack or z/OS image failure.

4.2.2 Virtual IP addressing

A virtual IP address (VIPA) is configured the same way as a normal IP address for a physical adapter, except that it is not associated with any particular interface. VIPA uses a virtual device and a virtual IP address that other TCP/IP hosts can use to select an z/OS IP stack without choosing a specific network interface on that stack. The virtual device defined for the VIPA is always active and never fails.

Dynamic VIPA (DVIPA) was introduced by SecureWay® Communications Server for OS/390® V2R8 IP to enable the dynamic activation of a VIPA as well as the automatic movement of a VIPA to another surviving z/OS image after a z/OS stack failure. There are two forms of Dynamic VIPA, both of which can be used for takeover functionality:

- Automatic VIPA takeover allows a VIPA address to move automatically to a stack (called a backup stack) where an existing suitable application instance is already active and allows the application to serve the client formerly going to the failed stack.
- Dynamic VIPA activation for an application server allows an application to create and activate VIPA so that the VIPA moves when the application moves.

Non-disruptive, immediate, automatic VIPA take back was introduced by IBM Communications Server (CS) for OS/390 V2R10 to move the VIPA back to where it originally belongs once the failed stack has been restored. This take back is nondisruptive to existing connections with the backup stack and the take back is not delayed until all connections with the backup stack have terminated (as was the case with CS for OS/390 V2R8 IP). New connections are handled by the new (original) primary owner, thereby allowing the workload to move back to the original stack.

4.2.3 Sysplex Distributor

Sysplex Distributor is designed to address the requirement of one single network-visible IP address for the sysplex cluster. Clients in the network receive the benefits of workload distribution and high availability across the sysplex cluster. With Sysplex Distributor, client connections seem to be connected to a single IP host even if the connections are established with different servers in the same sysplex cluster.

The example shown in Figure 4-3 illustrates the use of Sysplex Distributor to achieve workload balancing between two front-end CICS regions, PTA1 and PTA2, running on different LPARs. The URL used by the service requester applications contains the VIPA address of the distributing stack 9.12.4.68. The distributing stack resolves new connections for this VIPA address to one of the specific addresses used by the target servers (9.12.4.75 and 9.12.4.76).

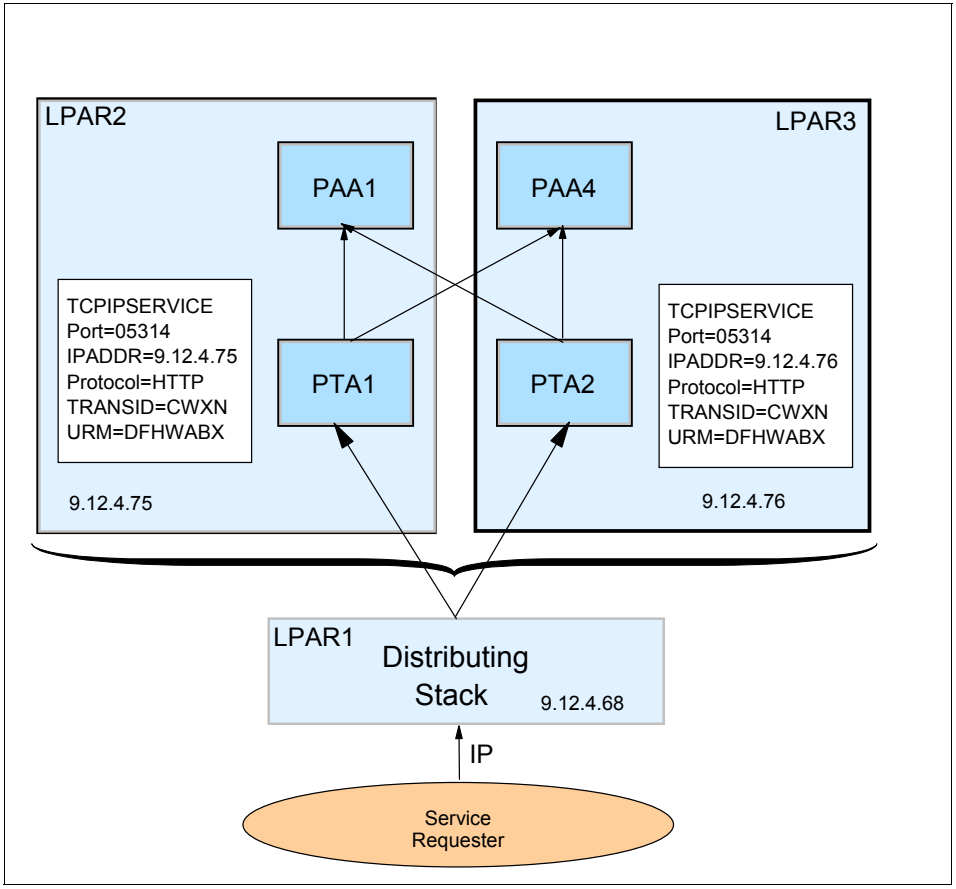


Figure 4-3 Sysplex Distributor example

Important: It is possible for the distributing stack (also known as the “Distributed DVIPA”) to also be a target stack (also known as the "unadvertised DVIPA").

As with TCP/IP port sharing, Sysplex Distributor also supports server-specific WLM recommendations for load balancing. The distribution of new connection requests can now be based on the actual workload of a target server. The new interface (SERVERWLM) indicates how well a specific server on a stack is meeting the goals of the WLM service class for the transactions that it is servicing (server-specific weights). This additional granularity enables the distributor to balance the workload more effectively.

Sysplex Distributor also takes into account information such as Quality of Service (QoS) and policy information provided by CS for z/OS IP’s Service Policy Agent. By combining this information with the information from WLM, the Sysplex Distributor has the unique ability to ensure that the best destination server instance is chosen for a particular client connection.

See Chapter 5, “Configuring CICS as a service provider” on page 103 for more details.

Sysplex Distributor has benefits over other load-balancing implementations:

- XCF links can be used between the distributing stack and target servers as opposed to LAN connections, such as an Open Systems Adapter (OSA).
- It provides a total z/OS solution for TCP/IP workload distribution.
- It provides real-time workload balancing for TCP/IP applications even if clients cache the IP address of the server (a common problem for DNS/WLM).
- It provides for take over of the VIPA by a backup system if the distributing stack fails.
- It allows for nondisruptive take back of the VIPA original owner to get the workload to where it belongs. The distributing function can be backed up and taken over.

It is possible to combine the use of Sysplex Distributor with TCP/IP port sharing for a high availability CICS service provider configuration. Then the Sysplex Distributor distributes requests across LPARS, and port sharing distributes requests across different CICS systems within an LPAR.

When CICS is hosting a Web service requester application, Sysplex Distributor can also be used to route requests to multiple instances of the service provider, so long as the service provider application runs within the same parallel sysplex

as the requester. If the service provider application does not run in the same parallel sysplex, there is the possibility of using a load balancing advisor to provide similar functionality.

For more information about TCP/IP load balancing techniques, refer to *Communications Server for z/OS V1R8 TCP/IP Implementation Volume 3: High Availability, Scalability, and Performance*, SG24-7341.

4.2.4 Security considerations

It is important to enable persistent TCP/IP connections when using SSL because it minimizes the cost of SSL handshaking. The SOCKETCLOSE attribute of the TCPIPSERVICE definition controls when a TCP/IP socket is closed. The default setting for the SOCKETCLOSE attribute is NO. Therefore, when a connection is made between a Web service client and CICS, by default CICS keeps the connection open until the Web service client closes the connection. You can set a value (in seconds) for the SOCKETCLOSE attribute if you want to close a persistent connection after the timeout period is reached.

When TCP/IP connections do not persist, or if the connection has timed out, it is still possible to avoid the impact of a full SSL handshake by reusing an SSL session ID. Session IDs are tokens that represent a secure connection between CICS and an SSL client. The session ID is created and exchanged between the SSL client and CICS during the SSL handshake. If a match is found, then an SSL session can be established without the impact of validating SSL certificates.

An SSL session ID may be retained by CICS in a local SSL cache, or can be shared across multiple CICS regions by using the sysplex session cache support provided by System SSL.

Sharing SSL session IDs across different CICS regions is particularly useful when Web service requests are being routed across a set of CICS regions using TCP/IP connection workload balancing techniques, such as TCP/IP port sharing or Sysplex Distributor. If the cache is shared between the CICS regions, the number of full SSL handshakes can be significantly reduced.

The SSLCACHE SIT parameter specifies whether CICS should use the local SSL cache or share a shared cache. When SSLCACHE=SYSPLEX is specified, an SSL session established with a CICS region on one system in the sysplex can be resumed using a CICS region on another system in the sysplex as long as the SSL client presents the session identifier obtained for the first session when initiating the second session. CICS uses the sysplex session cache support provided by the System SSL started task (GSKSRVR), an optional component of System SSL.

Figure 4-4 shows two CICS regions sharing access to an SSL sessions ID. A full SSL handshake is performed between a Web service requester and CICS region 1 running on LPAR1. The SSL session ID is stored by GSKSRVR in a data space so that when the same Web service client connects to CICS region 2 running on LPAR2, the SSL session can be resumed using the shared session ID.

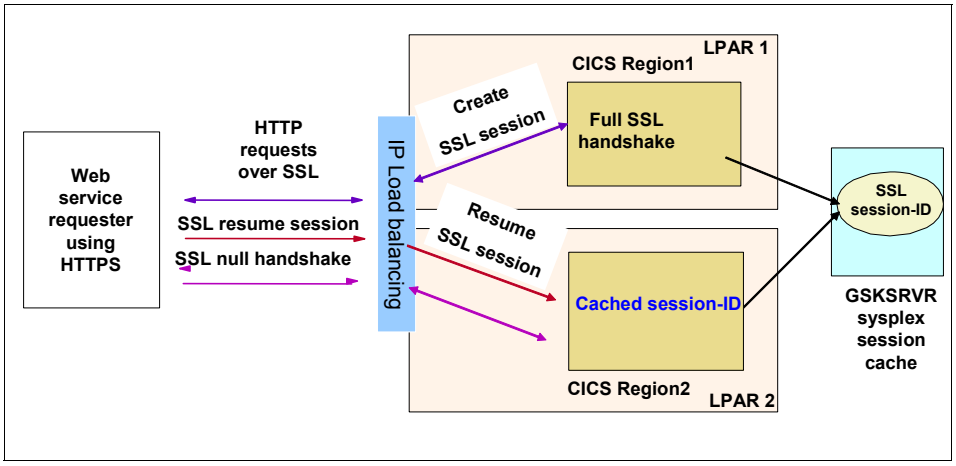


Figure 4-4 Sharing an SSL session ID across a sysplex

In order to use the sysplex session cache, each system in the sysplex must be using the same external security manager (for example, z/OS Security Server RACF®) and a user ID on one system in the sysplex must represent the same user on all other systems in the sysplex.

Recommendation: Sharing SSL session IDs across different CICS regions is very useful when Web service requests are being routed across a set of CICS regions. If the cache is shared between the CICS regions, the number of full SSL handshakes can be significantly reduced.

4.3 CICS as a service provider

In Figure 2-2 on page 26, we show how CICS processes a request to run a service provider application. The target business logic application may run in the same CICS region that receives the Web service request (the front-end region), or it may run in another region, for example, an AOR.

There are several advantages to running the target business logic program in a different region than the one that receives the Web service request:

- You can provide higher availability by having several regions that perform the same business function. If one of the regions fails, other regions of the same group can pick up the workload.
- You can implement workload balancing and workload separation.
- You are able to handle increasing workload by adding more CICS AORs.

There are two possible approaches to building a multiregion, front-end/back-end environment.

- The simplest approach is to use a distributed program link (DPL) to invoke the business logic program. All pipeline processing is done in the front-end region and the link to the business logic program is routed to an AOR. Separation of requests between different AORs may be achieved using workload separation based on different transaction IDs. We provide more information about the DPL approach in 4.3.2, “DPL routing” on page 88.
- It is also possible to route the entire pipeline and business logic processing to an AOR. This may be done by setting the transaction ID in the URIMAP and then dynamically routing this transaction. Setting the transaction ID in the URIMAP is necessary since you cannot change the definitions of the default CICS pipeline transaction CPIH. We provide more information about the transaction routing approach in 4.3.3, “Transaction routing” on page 92.

Routing of DPL requests has some advantages over the transaction routing approach:

- AORs that have not yet been migrated to CICS TS V3 can still be used for processing business application programs invoked as Web services.
- The DPL approach can be used regardless of whether the front-end regions and AORs are connected by MRO, ISC, or the new IP Interconnection protocol (IPIC). Transaction routing is supported only for MRO and ISC connections.
- It establishes a clearer definition and separation of the roles of the front-end regions and AORs.
- The DPL approach performs better than the transaction routing approach because it avoids the impact of routing part of the pipeline processing.

4.3.1 Determining the pipeline transaction code

The default transaction ID used for the Web services pipeline processing and business logic program is CPIH (or CPIQ when WebSphere MQ is used). For monitoring, accounting, or security reasons, you may want to run all processing of a given Web service request under a specific transaction ID.

This can be achieved by specifying a transaction ID on the URIMAP resource definition. The pipeline and business program processing is then performed under this transaction ID. A transaction ID may be specified on a URIMAP resource definition directly or by using the TRANSACTION parameter when running the Web services assist utilities (see “Using DFHLS2WS” on page 50).

You can also change the transaction ID of the target business logic program by modifying the contents of the context container DFHWS-TRANID in a message handler program. The effect of changing the transaction ID in a message handler is that the initial pipeline transaction runs with the transaction ID CPIH, but then the data mapping and business logic program runs in a separate task using the specified transaction ID.

If you are using DPL, you may also define a transaction ID on the remote program definition in the front-end region. This then allows the same transaction ID to be used in both the front-end region and the AOR. A mirror transaction definition specifying program DFHMIRs is required in the AOR.

4.3.2 DPL routing

In this section, we describe in more detail how you enable DPL routing. Figure 4-5 illustrates this model.

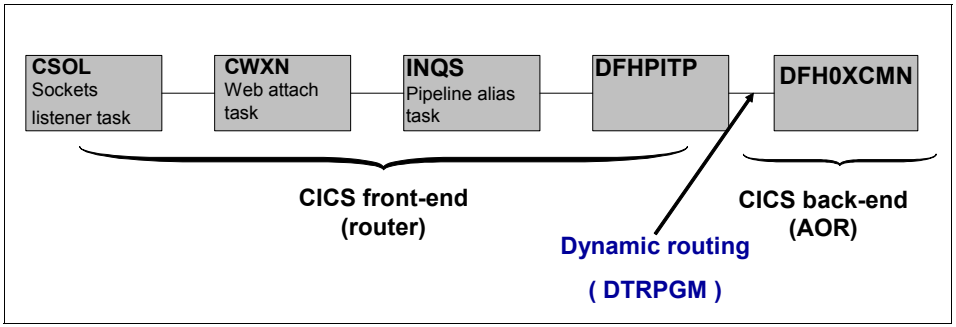


Figure 4-5 Web service provider - DPL routing

The processing sequence is summarized here:

- When the SOAP message arrives, the CICS-supplied sockets listener transaction (CSOL) attaches the transaction specified in the TRANSACTION attribute of the TCPIPSERVICE definition; by default, this will be the CICS-supplied Web attach transaction CWXN.
- CWXN uses the TRANSACTION attribute of the URIMAP definition to determine the name of the transaction that it should attach to process the pipeline; in our example, this transaction is INQS (by default, this will be the CPIH).
- The CICS-supplied SOAP message handler links to the program DFHPITP, which is the CICS-supplied target program for applications deployed with the CICS Web services assistant.
- After mapping the inbound service request (XML) to a COMMAREA or a container, DFHPITP links to the target business logic program, in our example, DFH0XCMN.

  The DPL routing is controlled by the program specified in the DTRPGM system initialization parameter. If CICSPlex SM is used for dynamic routing, DTRPGM=EUY9XLOP must be specified.
- The target business logic program DFH0XCMN runs on the AOR, which is chosen as the target server for the program link request.

Note: The pipeline processing runs entirely in the front-end region.

Figure 4-6 shows a variation of the DPL routing model.

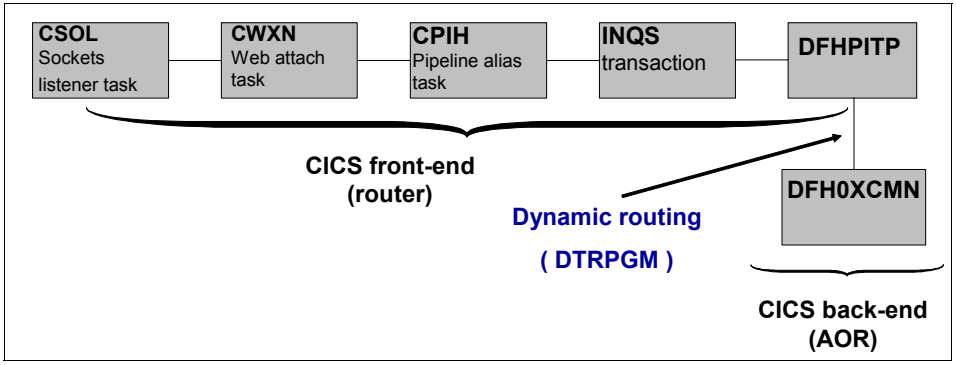


Figure 4-6 Web service provider - DPL routing

The processing sequence shown in Figure 4-6 on page 89 is the same as that shown in Figure 4-5 on page 88, with the exception that:

- The default pipeline transaction CPIH is started.
- A message handler specifies a new pipeline transaction ID by modifying the contents of the context container DFHWS-TRANID to INQS.
- A new task is started for the INQS transaction.

Resource definitions

Figure 4-7 shows the CICS resource definitions that are required for the DPL routing model.

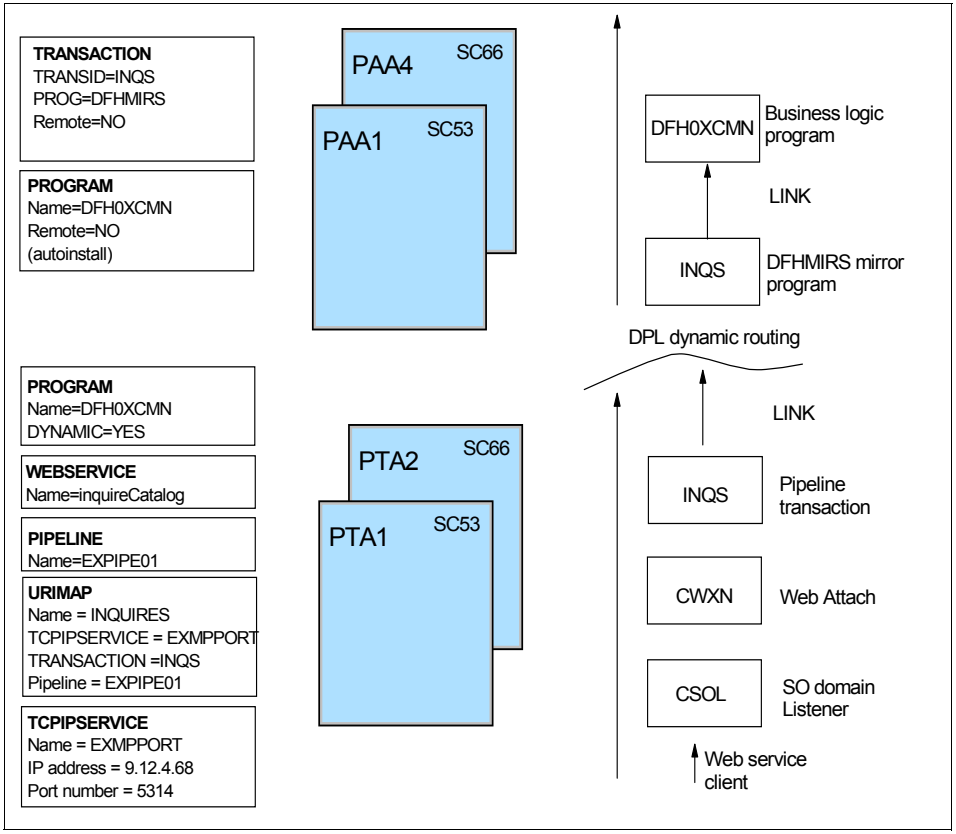


Figure 4-7 CICS resource definitions for the DPL routing scenario

The following resource definitions are made in the front-end regions PTA1 and PTA2:

- A TCPIPSERVICE specifying a shared port.
- A URIMAP specifying the pipeline transaction ID INQS.
- A PIPELINE definition.
- A WEBSERVICE definition.
- A PROGRAM definition specifying that the business logic program is remote and dynamic, and that the Transid of the remote transaction is INQS.

> **Note:** The dynamic routing program is also invoked if there is no program definition and program autoinstall is inactive, or, if program autoinstall is active and the autoinstall user program:
>
> - Installs a program definition that specifies DYNAMIC(YES)
> - Does not install a program definition

The following resource definitions are made in the back-end regions PAA1 and PAA4:

- The INQS transaction is defined as the CICS-supplied mirror program DFHMIRS.
- The target business logic program DFH0XCMN is defined as local.

Alternative ways of specifying the name of the remote transaction ID for a DPL are:

- You may use the TRANSID parameter on the LINK command. This requires that you are able to change the source of the application handler program (which is not the case if you use the Web services assistant).
- You may enable an XPCREQ (Program Control Request) global user exit in the front-end region. In this exit, you can change the transaction ID under which the remote program will run in the AOR.

> **Note:** We used the XPCREQ method to modify the mirror transaction because it was not possible to update the CICS-supplied program definition for DFH0XCMN in the front-end region. The XPCREQ exit that we used set the name of the mirror transaction ID in the AOR to the same value as the transaction ID of the invoking task (the pipeline transaction). However, unless you are also using a CICS supplied program, the simplest way to modify the mirror transaction is to update the Transid option on the program definition in the front-end region.

Regardless of the method that you use, we recommend that the transaction ID used for the pipeline processing in the front-end region is the same as the mirror transaction ID used in the AOR. This makes it easier to manage security, monitoring, accounting, and statistics for your Web service request processing.

4.3.3 Transaction routing

In this section, we describe in more detail how you enable dynamic transaction routing. Figure 4-8 illustrates this model.

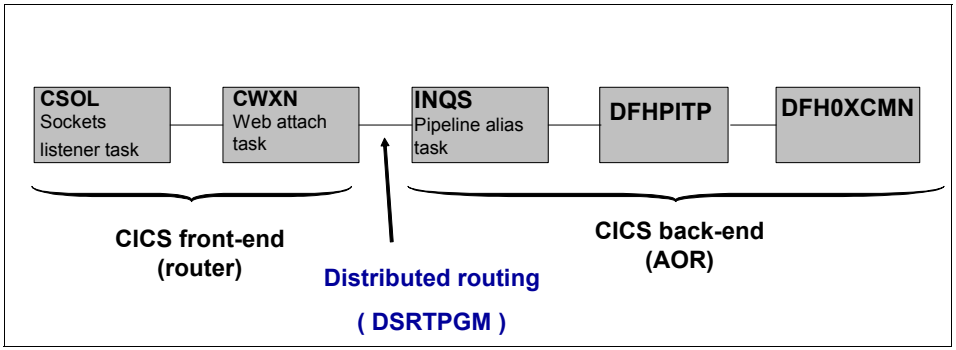


Figure 4-8 Web service provider - Transaction routing

The transaction that runs the pipeline and business processing is eligible for routing when the transaction is defined with DYNAMIC=YES.

The routing is controlled by the program specified in the DSRTPGM system initialization parameter. If CICSPlex SM is to be used for dynamic routing, DSTRPGM=EUY9XLOP must be specified.

The processing sequence is summarized below.

- When the SOAP message arrives, the CICS-supplied sockets listener transaction (CSOL) attaches the transaction specified in the TRANSACTION attribute of the TCPIPSERVICE definition; by default, this will be the CICS-supplied Web attach transaction CWXN.
- CWXN uses the TRANSACTION attribute of the URIMAP definition to determine the name of the transaction that it should attach to process the pipeline; in our example, this transaction is INQS (by default, this will be the CPIH).

- The pipeline transaction INQS is defined as remote and dynamic, so the transaction will be routed to a suitable AOR. The transaction routing is controlled by the program specified in the DSRTPGM system initialization parameter. If CICSPlex SM is used for dynamic transaction routing, DSRTPGM=EUY9XLOP must be specified.
- The INQS transaction is routed to the AOR and the CICS-supplied SOAP message handler links to program DFHPITP, which is the CICS-supplied target program for applications deployed with the CICS Web services assistant.
- After mapping the inbound service request (XML) to a COMMAREA or a container, DFHPITP links to the target business logic program, in our example, DFH0XCMN.
- The target business logic program DFH0XCMN runs on the AOR.

Note: The pipeline processing runs entirely in the back-end region (AOR).

Figure 4-9 shows a variation of the transaction routing model.

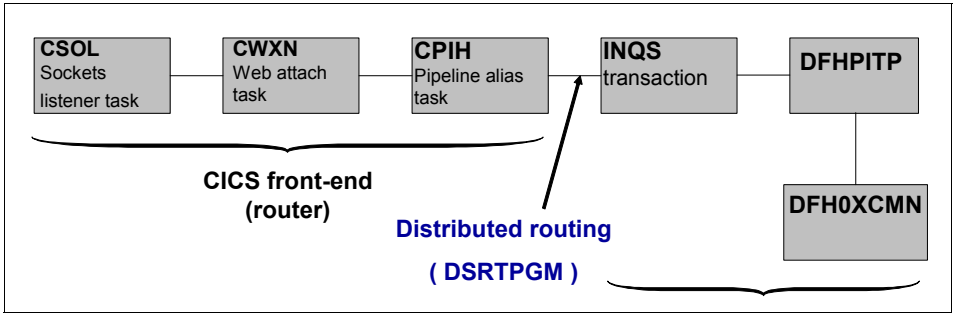


Figure 4-9 Web service provider - Transaction routing

The processing sequence shown in Figure 4-9 is the same as that shown in Figure 4-8 on page 92, with the exception that:

- The default pipeline transaction CPIH is started.
- A message handler specifies a new pipeline transaction ID by modifying the contents of the context container DFHWS-TRANID to INQS.
- A new task is started for the INQS transaction. INQS is defined as remote and dynamic so the transaction is routed to a suitable AOR.

Note: The pipeline processing runs partly in the front-end region and partly in the back-end region (AOR).

Resource definitions

Figure 4-10 shows the CICS resource definitions that are required for the dynamic transaction model.

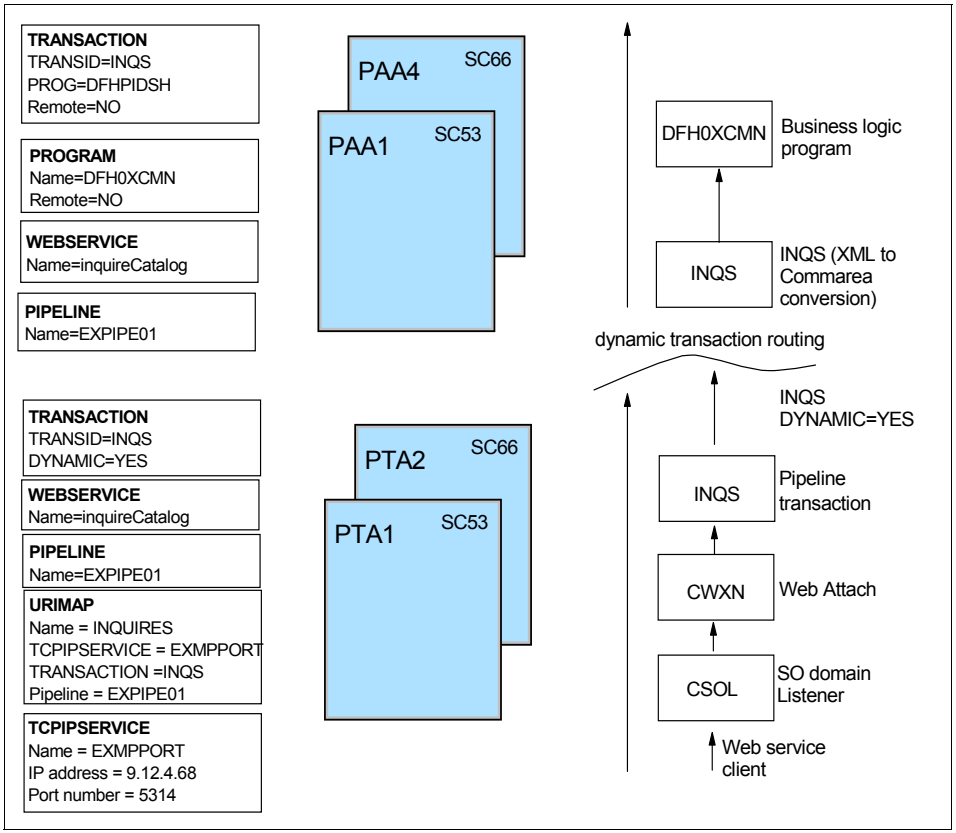


Figure 4-10 CICS resource definitions for transaction routing scenario

The following resource definitions are made in the front-end regions PTA1 and PTA2:

- A TCPIPSERVICE specifying a shared port
- A URIMAP specifying the pipeline transaction ID INQS
- A PIPELINE definition
- A WEBSERVICE definition
- A TRANSACTION definition specifying that the transaction is remote and dynamic

The following resource definitions are made in the back-end regions PAA1 and PAA4:

- PINELINE and WEBSERVICE resource definitions are required in the back-end region, since the pipeline processing is done here.
- The INQS transaction is defined as the CICS-supplied SOAP message handler program DFHPIDSH.
- The target business logic program DFH0XCMN is defined as local.

4.4 CICS as a service requester

For different reasons, you might want to run Web service requester applications in a number of dedicated CICS systems, as shown in Figure 4-11.

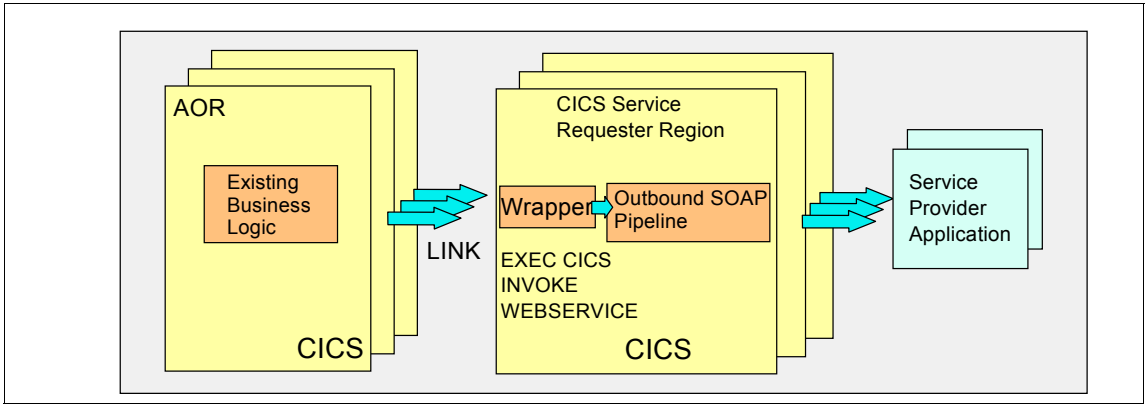


Figure 4-11 Workload balancing across service requester regions

Business logic programs running in AORs do not invoke Web services directly; instead, they link to a wrapper program that makes outbound Web service calls using the EXEC CICS INVOKE WEBSERVICE command. The wrapper program runs in a CICS system that is dedicated to hosting service requester applications.

In Chapter 5, “Configuring CICS as a service provider” on page 103, we show how we configured the catalog manager application to run in a configuration similar to Figure 4-11.

Some advantages of using CICS service requester regions are:

- Business logic programs running in CICS systems that are not using CICS TS V3.1 or higher can make Web service calls indirectly through the CICS service requester region.
- You do not need to define Web services pipelines in all AORs. Outbound SOAP pipelines need only to be created in a limited set of CICS service requester regions.
- You may not want the AORs to have access to external resources through TPC/IP.
- The Distributed Program Links from AORs can be workload managed by CICSPlex SM across a set of CICS service requester regions, thus providing workload balancing and higher availability in the event of CICS region failures.
- It is always good practice to clearly distinguish the roles of different CICS regions.

If there is more that one instance of the Web service provider, it is possible to distribute requests across the service provider instances. One solution could be to dedicate CICS service requester regions to make service requests to specific instances of the service provider application. Another solution is to workload balance requests between the service provider instances.

4.4.1 DPL routing

In the configuration shown in Figure 4-11 on page 95, DPL routing can be used to dynamically route program link requests from an AOR to CICS service requester regions. The routing can be managed by CICSPlex SM. Specific mirror transaction IDs can be assigned to the link requests, and workload separation can also be configured if required.

Business logic program considerations

The business logic program is not Web service aware; it simply links to the wrapper program using a standard CICS COMMAREA or container.

Wrapper program considerations

The wrapper program is Web service aware. It invokes the Web service based on input from the business logic program.

The target location of the service provider application is specified in the *soap:location* from the *wsdl:port* specified in the WSDL of the service. This location address is used by the DFHWS2LS Web services assistant utility and is stored in the wsbind file for the service. The wsbind file is used by CICS at the time of processing the outbound Web service request.

The wrapper program can override the target location of the service provider with the URI option of the EXEC CICS INVOKE WEBSERVICE command.

The wrapper program may implement some sort of balancing between multiple instances of the Web service provider. This can be as simple as round-robin routing. It can also handle a situation where one of the service providers is not responding. In this case, a retry can be made to an alternative service provider application. In 6.2, “Using multiple Web service providers” on page 169, we show how we configure the catalog manager application dispatchOrder service in this way.

It is important to make sure that the wrapper program does not wait indefinitely for the service provider application to respond, as this can have serious consequences for both the service requester regions and the AORs.

Resource definitions

The wrapper program is defined as REMOTE and DYNAMIC in the AORs. An alias mirror transaction ID can also be specified in the remote program definition.

In the CICS service requester regions, you must define the WEBSERVICEs that are invoked and the service requester PIPELINEs. These regions must have access to the wsbind and WSDL directories.

If the CICS service requester region is at the CICS TS V3.2 level, you can define how long a service requester application should wait for a reply from the service provider application. The PIPELINE resource has a new RESPWAIT attribute that determines how many seconds CICS waits for a reply.

If you do not set a value for the RESPWAIT attribute, or if the CICS system is at a lower level than CICS TS V3.2, either the default timeout for the transport protocol or the deadlock timeout (DTIMOUT) for the transaction is used.

- The default timeout for HTTP is 10 seconds.
- The default timeout for WebSphere MQ is 60 seconds.

If the value for the deadlock timeout for the transaction is less than the default for either protocol, the timeout occurs with the CICS dispatcher (DTIMOUT).

4.5 Monitoring availability and performance

In order to achieve the highest possible availability for a CICS Web services application, it is important to be able to preempt problems before they occur, using threshold monitoring, for example, and to act immediately if a problem does occur. Some problems can be handled automatically, but others may need to be handled manually.

The following tools can be used to monitor a CICS Web services infrastructure:

- CICSPlex SM for operations and monitoring
- IBM Tivoli OMEGAMON® XE for CICS for a more detailed analysis of Web services in CICS and integration with the Tivoli Enterprise Portal (TEP)
- IBM Tivoli Composite Application Manager for SOA (ITCAM for SOA) for monitoring Web services across different runtime environments

In this book, we focus on using CICSPlex SM as the primary operations and monitoring tool. However, in 7.6, “Workload management and monitoring scenarios” on page 212, we also show how OMEGAMON XE for CICS and ITCAM for SOA can provide detailed information about the performance and health of CICS Web services.

4.5.1 CICSPlex SM monitoring

CICSPlex SM has three main functions for monitoring CICS:

- System Availability Monitoring (SAM) can monitor the overall health of the CICS system.
- MAS Resource Monitoring (MRM) is used for monitoring specific resources within CICS.
- CICS monitoring can be used for monitoring the workload.

In order to deal with planned outages for the CICS systems, you can create definitions (PERIODEFs) in CICSPlex SM to define when a CICS system must be active. This stops alerts from being issued when a system is brought down for maintenance or during an “out of hours” period.

System availability monitoring

SAM can be used for monitoring the following situations:

- CICS is not running.
- CICS is short on storage.
- CICS is taking a system dump.

- CICS is taking a transaction dump.
- CICS is in a stall condition.

MAS resource monitoring

MAS resource monitoring (MRM) is used for monitoring CICS resources, such as transactions, programs, or connections.

There are a number of resources that are critical to the system. In the event that one of these resources is in a state that may cause a problem for the application, it is possible to configure CICSPlex SM to raise an event. It may also be possible to correct the problem automatically using the operations capability of CICSPlex SM.

The following are examples of how MRM can be used to minimize the impact of resource failures of service availability:

- If a TCPIPSERVICE is closed, you can configure CICSPlex SM to raise an event and to try to open the TCPIPSERVICE.
- If a CICS connection is not acquired, an event can be raised and CICSPlex SM can try to acquire the connection.
- If a VSAM file is closed, an event can be raised and CICSPlex SM can try to open the file.
- If the number of transaction dumps exceeds 25, an event can be raised.

CICS monitoring

CICS monitoring can monitor the throughput and performance of the system. The following are good examples of what to monitor:

- If the transaction rates drop below one transaction per second, an event can be raised. This indicates that no requests are being received, for example, there may be a network problem.
- An event can be raised if the average transaction response time exceeds one second.
- An event may also be raised if the transaction rates exceed 500 per second. This indicates an exceptionally high number of requests are being received that may require additional resources.

4.5.2 Protecting the system

When the processing within a CICS system is being significantly impacted by application or infrastructure problems, the systems programmer is faced with two objectives, which are often difficult to manage at the same time. First, it is necessary to maintain service availability and this may require a CICS region to be recycled. Second, it is necessary to identify the root cause of the problem.

The following techniques can be used to protect service availability while at the same time collecting the information that will be required for diagnosing the cause of the problem.

Automatic recycling of CICS

In situations where CICS detects a severe problem but is unable to resolve it automatically, it is often best to recycle the CICS region without delay using System Automation or the CICS Automatic Restart Manager (ARM) capability. Whenever possible, a system dump should be taken for later problem determination.

An example of this situation is when the CICS Temporary Storage dataset becomes full. Setting a SYSDUMPCODE for dumpcode TS1311 informs CICS to take a system dump and abend. The CICS region can then be automatically restarted without manual intervention.

Transaction classes

In order to avoid problems with a single service from affecting other services, transaction classes can be used to isolate a group of services. Web services can be grouped into a number of transactions classes if different transaction IDs are associated with specific service requests. Each transaction class is set to the desired maximum number of concurrently active service requests. If one transaction class starts queuing transactions, this will not effect Web services defined in other transaction classes.

The PURGETHRESH attribute of a TRANCLASS definition is used to set the limit of the queue for the transaction class. It can be used to limit the number of concurrent tasks that are created for a particular Web service. This can be useful in avoiding queuing more requests than we expect to be able to process within the time the requester will wait for a response. Otherwise, it is possible to end up processing requests from requesters that have already timed out.

It is important to avoid a situation in which CICS reaches the MAXTASK condition, preventing CICS from attaching any new tasks. To prevent this, the sum of maximum active tasks for all transaction classes should be set lower that the MAXTASK setting.

Short on storage

To allow CICS to purge tasks in the case of a short-on-storage condition, you should define transactions to CICS with the DTMIOUT=xx and SPURGE=YES attributes. This significantly increases the possibility of CICS being able to recover from a short-on-storage condition. Note that:

- SPURGE=YES specifies whether CICS can purge the task as a result of the expiration of a deadlock timeout (DTIMOUT) delay interval.
- DTIMOUT=xx specifies the interval in seconds after which CICS initiates an abnormal termination of the task when the task is suspended (inactive).

5

Configuring CICS as a service provider

In this chapter, we describe how CICSPlex SM was used to define and manage the required CICS definitions to run the Web services application. We discuss these topics:

- The application
- The configuration
- The CICS configuration changes required
- The TCP/IP configuration changes required
- Defining the CICSPlex SM routing definitions
- Defining the CICS resources using CICSPlex SM
- Using CICSPlex SM to manage the environment

5.1 The application

The application we selected for running the scenarios is the sample catalog application shipped with CICS TS V3.2. See Chapter 14, “The CICS catalog manager sample application”, in *Web Services Guide*, SC34-6838, for further information.

5.1.1 Requirements

We had to consider two areas of high availability:

- We wanted to be able to distribute inbound Web services requests across more that one front-end region running on more than one z/OS image. Which high availability solution is best for obtaining this goal?
- We wanted to be able to distribute business logic processing across more that one back-end region running on more than one z/OS image.

5.1.2 The configuration

We decided to use the Sysplex Distributor/Dynamic VIPA approach for distributing incoming requests between the front-end regions. It is capable of balancing IP connection requests in a z/OS IP domain, which allows us to run one or more CICS routing regions on each of the z/OS images that we use. Furthermore, this approach is utilizing MVS WLM services to control the workload based on the health of registered CICS routing regions.

We also decided to set up CICSPlex SM to distribute dynamically the business logic processing across the CICS application regions. We used CICSPlex SM to do dynamic DPL routing of the business logic program DFH0XCMN.

Incoming Web service requests get processed by the CICS routing regions PTA1 and PTA2. Each routing region can then utilize one of the CICS application regions to run the business logic program.

We *did not* use a message handler in the pipeline to change the transaction ID for our Web service; instead, we set the transaction IDs in the URI mapping definitions.

We ran both the front-end pipeline transaction and the back-end business transaction under the same transaction code, thus enabling better security, accounting, monitoring, and statistics gathering.The business logic program was defined as DYNAMIC=YES in the routing regions, and autoinstalled in the application regions.

We have configured a high availability scenario for our Web services application, as shown in Figure 5-1 on page 106.

5.1.3 The CICS configuration

The configuration to run the application is set up as follows:

- Two existing CICS front-end regions SCSCPTA1 and SCSCPTA2 are used.
- Two existing CICS back-end regions (AOR) SCSCPAA1 and SCSCPAA4 are used.
- SCSCPTA1 and SCSCPAA1 run on LPAR SC53.
- SCSCPTA2 and SCSCPAA4 run on LPAR SC66.
- Each CICS front-end region listens for the HTTP requests using the same VIPA port for fail over and workload balancing.
- All CICS to CICS connections are set up using IP Connections introduced with CICS TS V3.2.
- Routing to the business application is done using DPL.

Figure 5-1 illustrates the configuration.

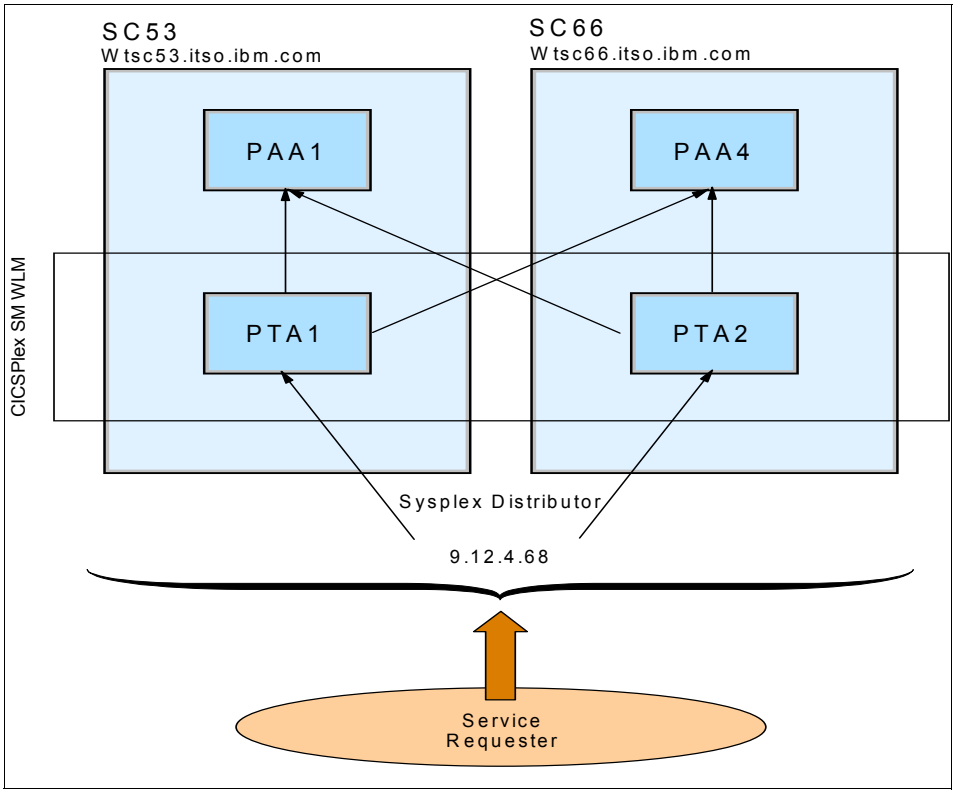


Figure 5-1 Catalog application configuration

The following CICS system groups are defined to CICSPlex SM:

- ICAT: Contains all four regions.
- ICATTORS: Contains SCSCPTA1 and SCSCPTA2.
- ICATAORS: Contains SCSCPAA1 and SCSCPAA4.

The definitions used to create the infrastructure are described in more detail later in this chapter.

The program that is to be linked to dynamically is DFH0XCMN. Unfortunately, the only remote attribute that can be applied to programs starting with DFH is DYNAMIC=YES. In order to be able to make the transaction in the front-end region and the back-end region run under the same transaction ID, we created a global user exit program to run off XPCREQ in the front-end region. This exit sets the remote transaction ID equal to the current transaction ID.

5.2 TCP/IP definitions

During the testing of the sample application, two methods were used to access the CICS regions through TCP/IP. TCP/IP port sharing was used when both CICS front-end regions were running on the same LPAR. To enable TCP/IP port sharing, we made the following changes to the TCP/IP profile:

```
5314TCP SCSCPTA1 SHARPORT;
5314TCP SCSCPTA2 SHARPORT;
```

The final configuration, as shown in Figure 5-1 on page 106, was set up to use Dynamic VIPA and Sysplex Distributor so that Web Services requests are distributed across the CICS TOR regions on multiple LPARS.

An entry was placed in the DNS for Catalog.itso.ibm.com, and entries were placed into the dynamic VIPA definitions on SC66 and SC53. The VIPA definitions are detailed in Figure 5-2 and Figure 5-3.

```
VIPADYNAMIC
 VIPADEFINE MOVE IMMEDIATE 255.255.255.0  9.12.4.68
 VIPADISTRIBUTE DEFINE DISTMETHOD BASEWLM 9.12.4.68
    PORT 5314
    DESTIP 10.1.100.66 10.1.100.53
 ENDVIPADYNAMIC
```

Figure 5-2 VIPA definition on SC66

```
 VIPADYNAMIC
 VIPADEFINE MOVE IMMEDIATE 255.255.255.0  9.12.4.68
 VIPADISTRIBUTE DEFINE DISTMETHOD BASEWLM 9.12.4.68
    PORT 5314
    DESTIP 10.1.100.66 10.1.100.66
 ENDVIPADYNAMIC
```

Figure 5-3 VIPA definition on SC53

5.3 CICS definitions

In this section, we describe the CICS definitions for the various options.

5.3.1 System initialization options

With the introduction of CICS Transaction Server V2.2, a new SIT option CPSMCONN was introduced so that PLT did not have to be used to activate CICSPlex SM.

There are three CPSMCONN options:

- CPSMCONN=CMAS: Indicates that the CICS region is a CMAS.
- CPSMCONN=LMAS: Indicates that the CICS region is a MAS.
- CPSMCONN=WUI: Indicates that the CICS region is being used as a CICSPlex SM Web User Interface (WUI).

CICS Web Services transactions are running under a non-terminal related task in the front-end regions. To enable CICSPlex SM balancing of dynamic program link to the business application program in the back-end regions, the following SIT option needs to be added to all front-end regions:

```
DTRPGM=EYU9XLOP
```

5.3.2 CICS - CICSPlex SM initialization options

As part of CICS - CICSPlex SM initialization, CICS reads the datasets referred to by EYUPARM to obtain the CICS/CICSPlex SM startup options.

There is one mandatory parameter, CICSPLEX(name), which indicates the name of the CICSplex to which the local MAS is to be associated.

Tip: The following parameters are recommended when using Business Application Services (BAS) or Workload Management (WLM):

- MASPLTWAIT(YES): This is used to prevent any CICS applications from running and users signing on until CICSPlex SM has initialized and completed installing all the resources for that CICS region from BAS.
- The length of time that is allowed for CICSPlex SM initialization is determined by the MASINITTIME parameter. The default is 10 minutes. You should monitor the length of time that it takes for your CICS systems to initialize and install their BAS definitions, and where necessary, increase the MASINITIME.

5.3.3 Catalog application definitions

The following resources must be defined to the back-end regions:

- File EXMPCAT
- The transactions INQC, INQS, and UPDT all refer to the program DFHMIRS

The following resources must be defined to the front-end regions:

- TCPIPService EXMPPORT.
- The transactions INQC, INQS, and UPDT all referring to program DFHPIDSH.
- Program DFH0XCMN. Must be defined with dynamic routing YES.
- Pipeline EXPIPE01.

The TCPIPService may have to be different for each TOR if the TCP/IP address or port numbers are different.

5.3.4 CICS connection definitions

The following definition must be made for all regions:

TCPIPservice DFHIPCI used by IP connection. These must be different for each region since they specify the TCP/IP address and port number.

The following definition must be made for the front-end regions:

IP connection pointing to the back-end regions. The same definitions may be used for all front-end regions.

The following definition must be made for the back-end regions:

IP connection pointing to the front-end regions. The same definitions may be used for all back-end regions.

5.4 CICSPlex SM WUI menus

The CICSPlex SM WUI was used for the creation and updating of all the CICS resources used by the sample application. All of the CICSPlex SM WUI menus referred to in this chapter are shown in Figure 5-4 on page 110 through Figure 5-8 on page 114.

Figure 5-4 CICSPlex SM WUI main menu

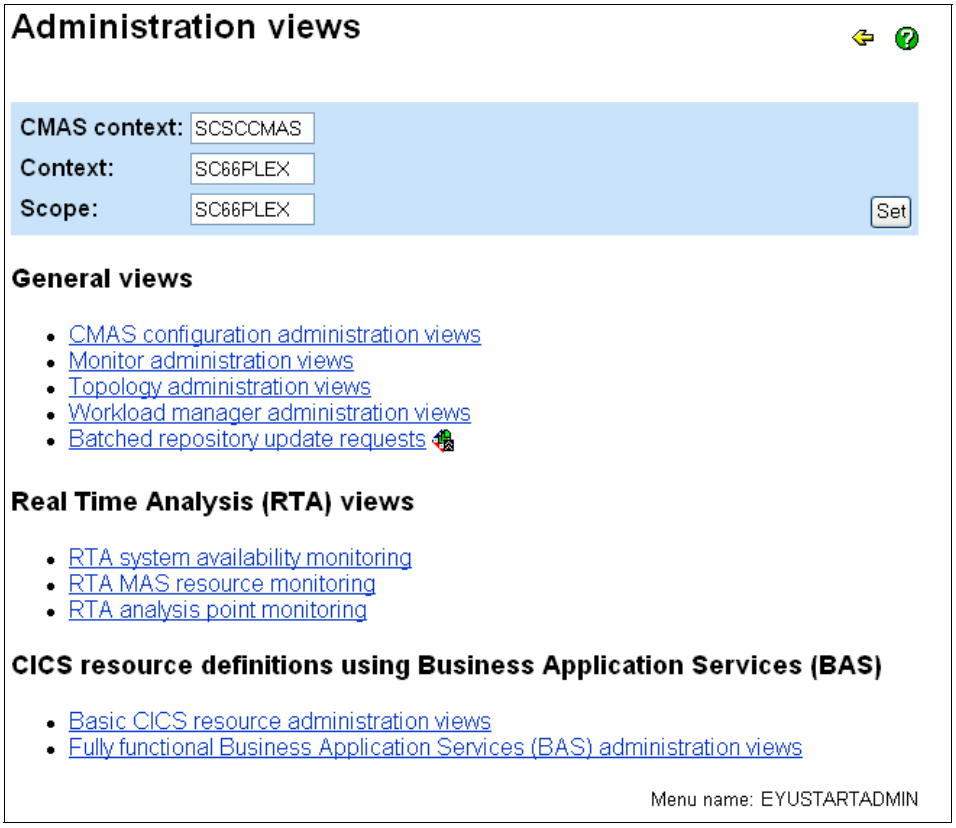


Figure 5-5 Administration view

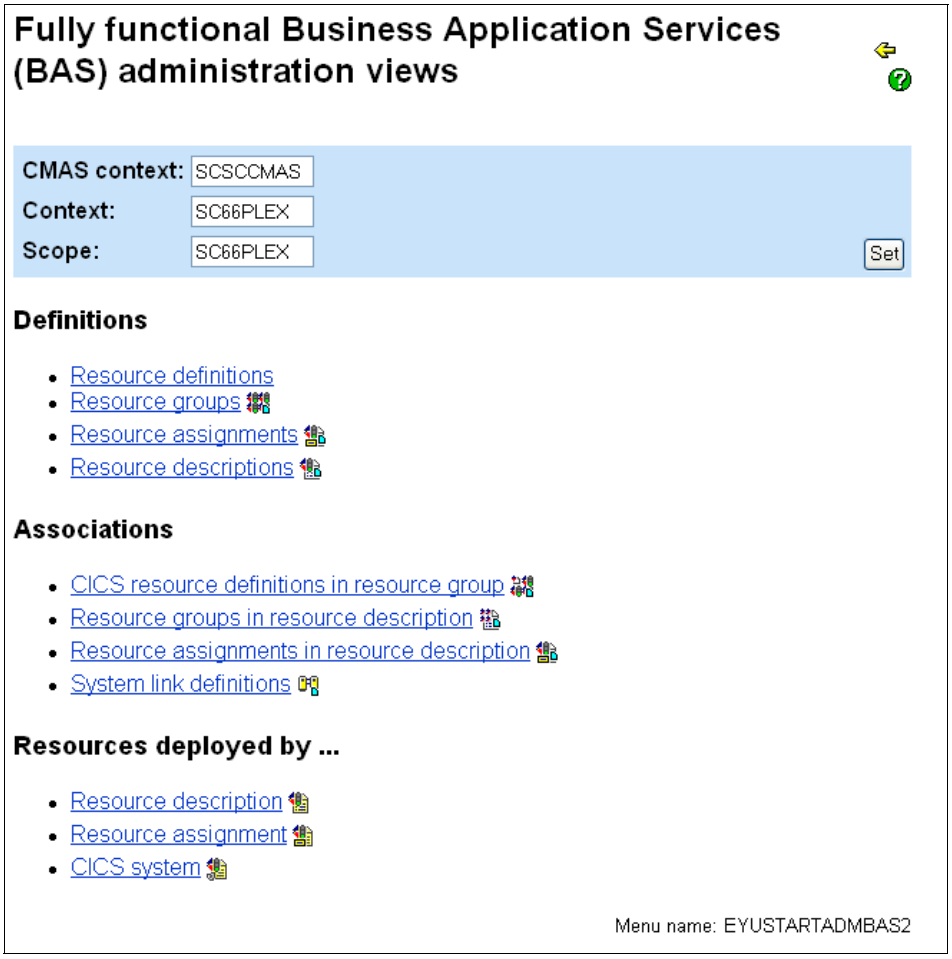


Figure 5-6 Fully functional BAS administration views

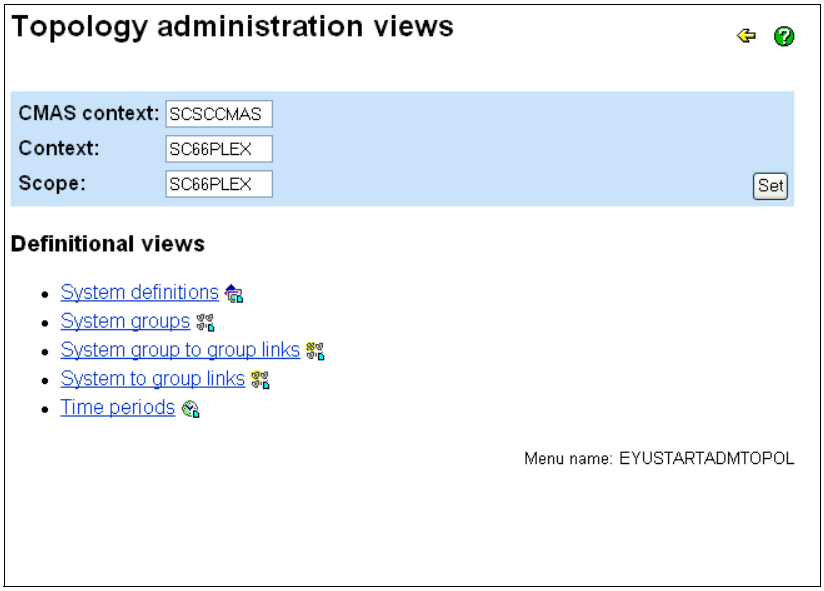


Figure 5-7 Topology administration views

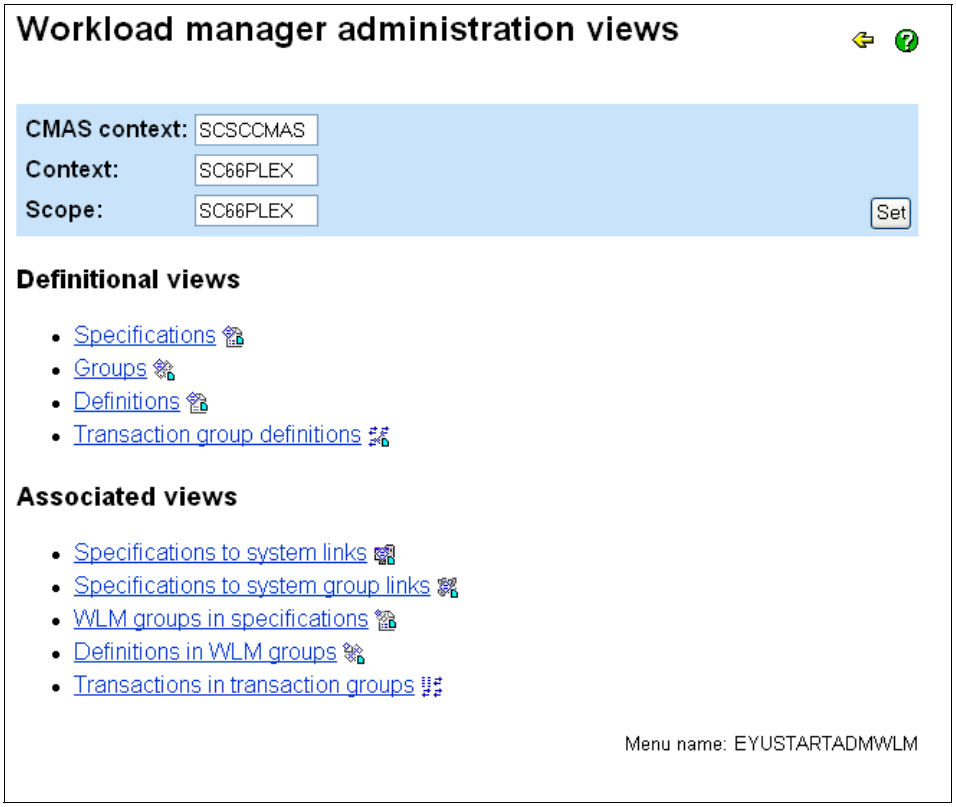


Figure 5-8 Workload manager administration views

5.5 CICS system definitions

In this section, we describe the CICS system definitions needed.

5.5.1 CICS region definitions

The CICS systems used to run the sample application were already defined to CICSPlex SM. All the CICS region definitions needed to be updated so that BAS will install the resource definitions.

To update the CICS region definitions from the main menu (Figure 5-4 on page 110), select **Administration views** → **Topology administration views.** → **System definitions** to display the view shown in Figure 5-9.

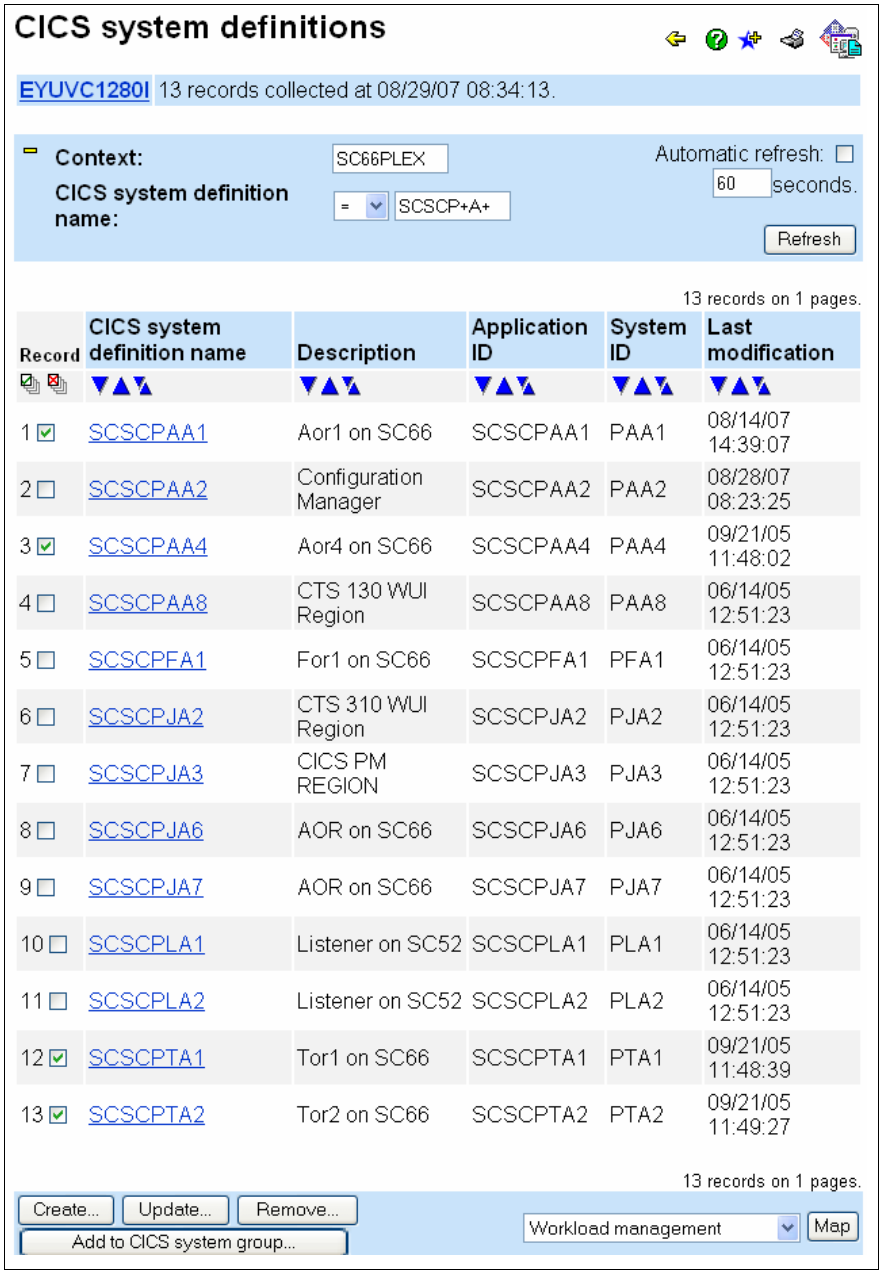


Figure 5-9 CICS system definition

After selecting the CICS region to be updated, click **Update** to update the CICS system definition (Figure 5-10).

The CICS system view has a large number of options. The BAS options are the only ones that need to be updated. These can be found at the bottom of the CICS system definition. Update the Install BAS resources option to COLDONLY.

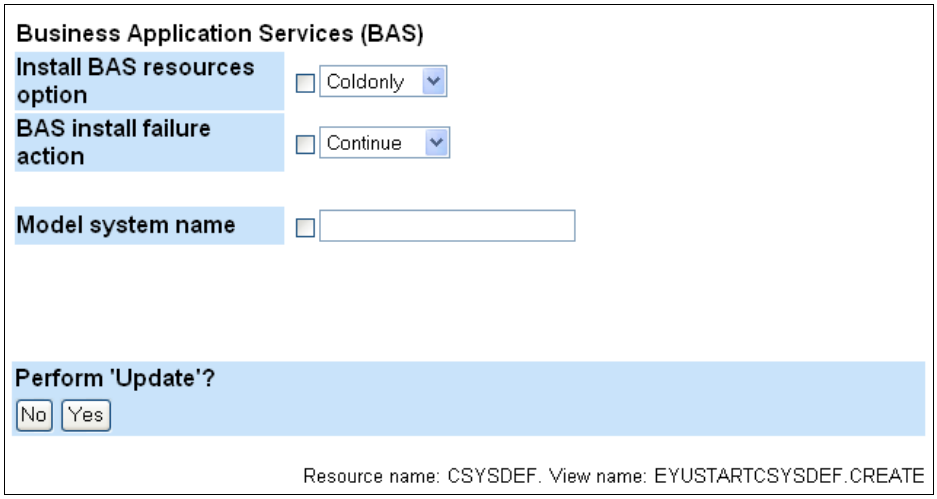


Figure 5-10 CICS system definition - BAS options only

Click **Yes** to update the CICS System definition.

All four CICS systems must be changed in this way.

Recommendation: Use COLDONLY so that the installation of resources works the same way as resources installed from the CICS CSD, and so there are no changes to the CICS resources upon a warm start.

5.5.2 System group definitions

To define CICS system groups from the main menu (Figure 5-4 on page 110), select **Administration views** → **Topology administration** → **System groups** → **Create** to display the window shown in Figure 5-11.

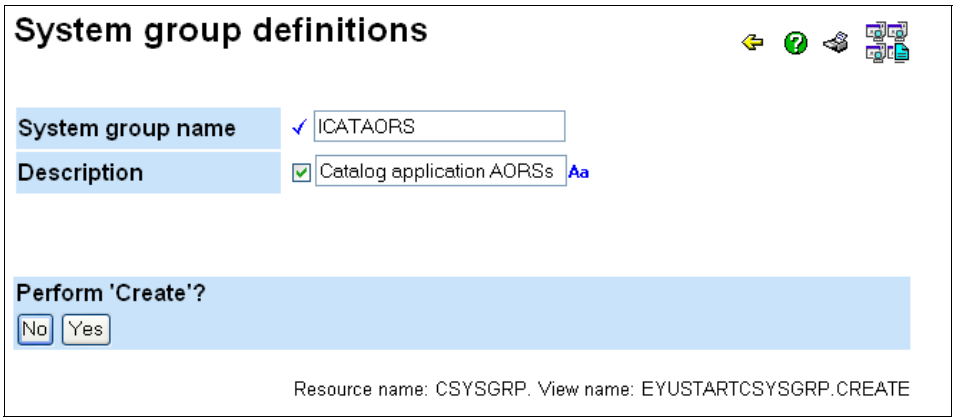


Figure 5-11 Creating a system group definition

Enter the system group name and descriptions. Click **Yes** to create the new system group definition.

Follow the same sequence of steps to create CICS System Groups ICATTORS and ICAT.

Adding CICS systems to system groups

In this scenario, we will use three system groups:

- ICAT: Contains all four regions.
- ICATTORS: Contains SCSCPTA1 and SCSCPTA2.
- ICATAORS: Contains SCSCPAA1 and SCSCPAA4.

To add CICS systems to system groups from the main menu (Figure 5-4 on page 110), select **Administration views** → **Topology administration views** → **CICS system definitions** to display the window shown in Figure 5-12.

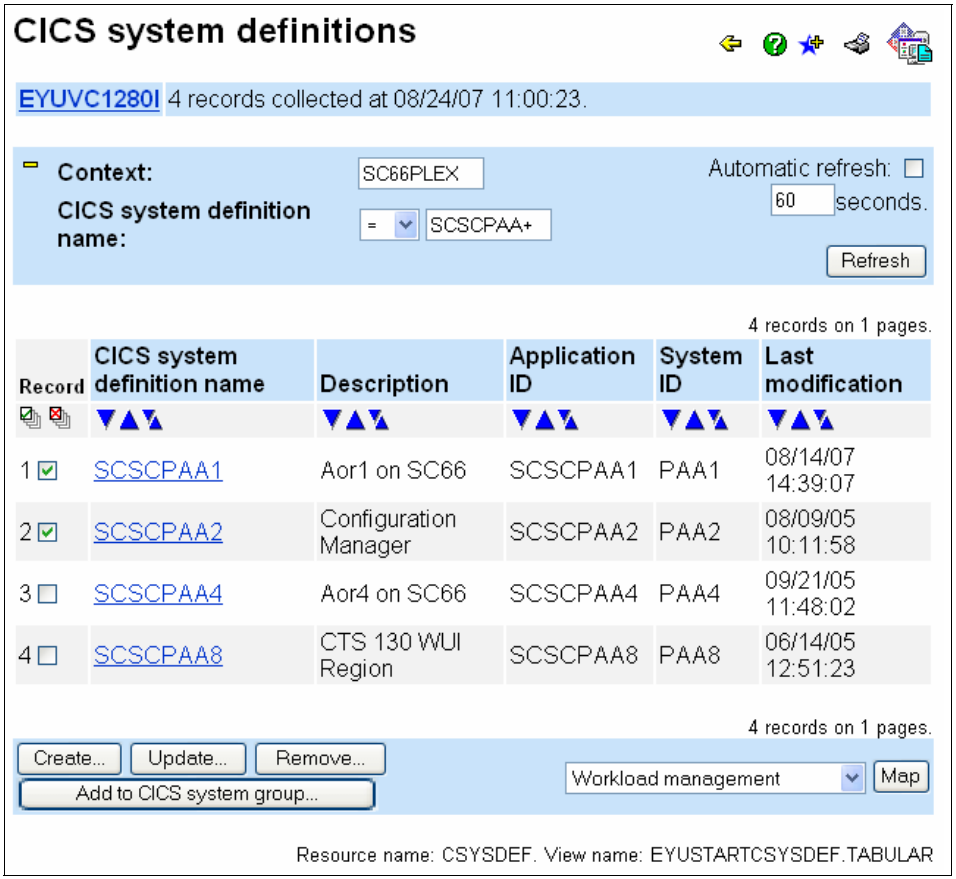


Figure 5-12 CICS system definitions

Select the CICS systems that you want to add and click **Add to CICS system group**. The view shown in Figure 5-13 is displayed.

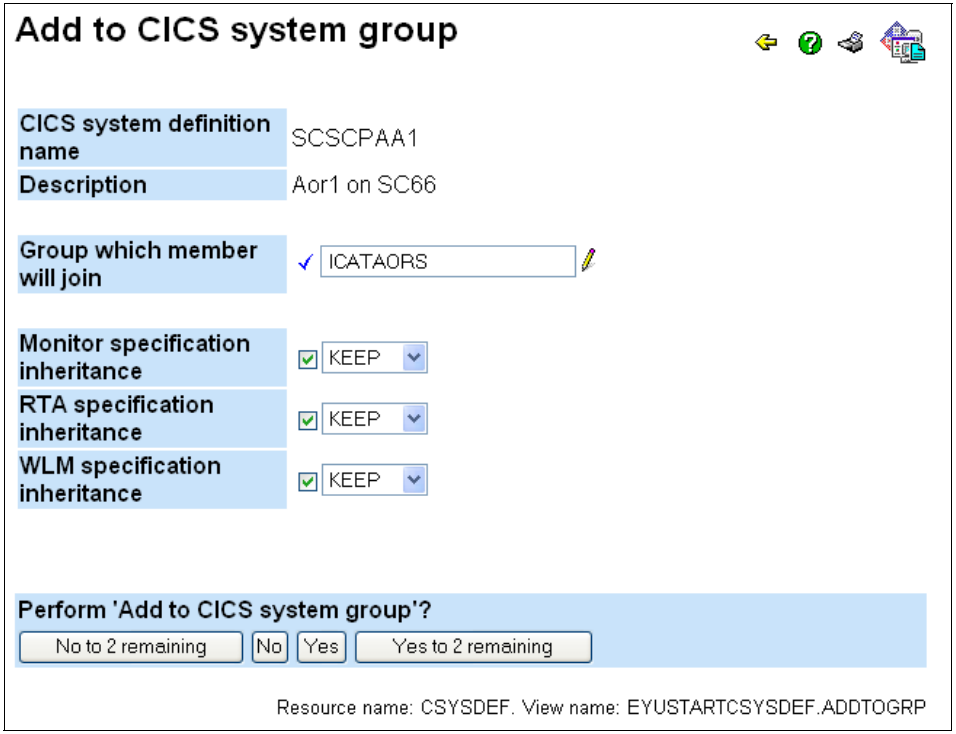


Figure 5-13 Add to CICS system group

Enter the name of the CICS group the selected CICS systems are to join. Leave Monitor, RTA, and Workload inheritance as KEEP. Click **Yes to 2 remaining** to add both systems selected in the previous view.

Afterwards, add SCSCPTA1 and SCSCPTA2 to CICS System Group ICATTORS.

Adding CICS systems groups to CICS system groups

To add CICS systems to another CICS system group from the main menu (Figure 5-4 on page 110), select **Administration views** → **Topology administration views** → **System groups**.

Select the two CICS system groups ICATAORS and ICATTORS and click **Add to CICS system group** (Figure 5-14).

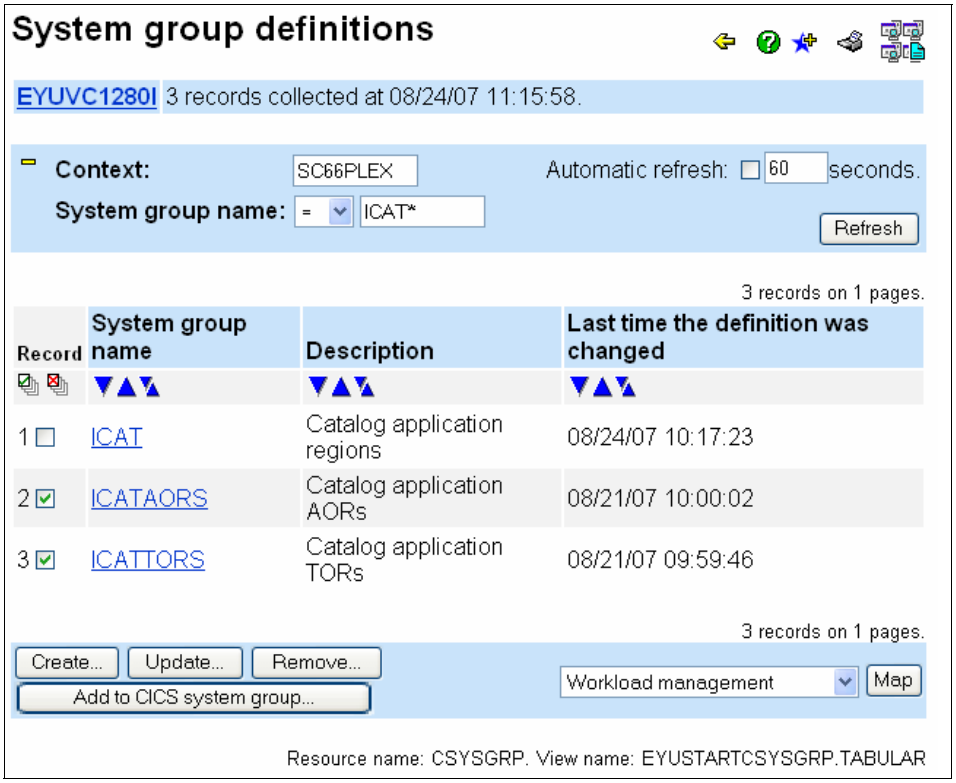


Figure 5-14 System group view

Enter the name of the CICS system group the selected system groups are to join (Figure 5-15). Leave Monitor, RTA, and Workload inheritance as KEEP. Click **Yes to 2 remaining** to add both systems groups selected in the previous view.

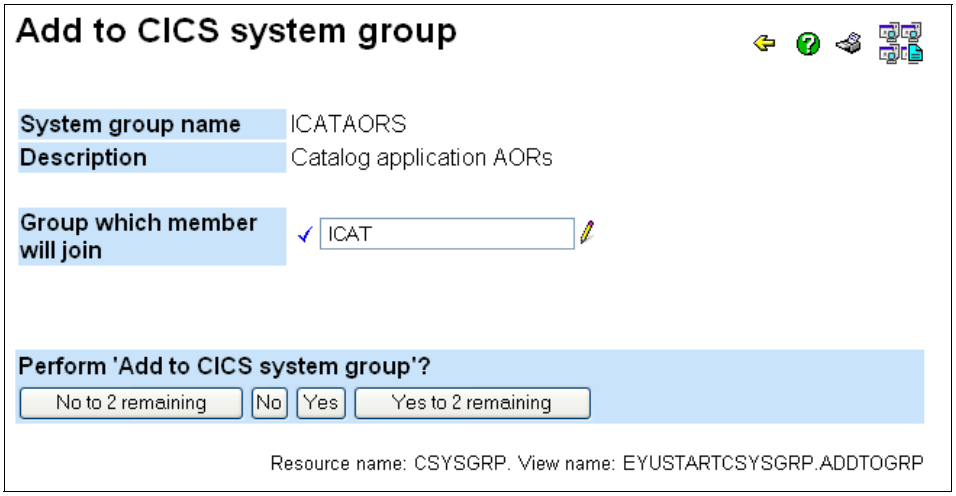


Figure 5-15 Adding a CICS system group to an existing CICS system group

5.5.3 Preparing to use Business Application Services

In order to be able to use Business Application Services (BAS) to define resources to CICS, we have to set up some definitions to link resource definitions to CICS systems. This is done using resource groups and resource descriptions.

Setting up resource groups

Resource groups are used for grouping resources into groups. These groups can then be added to resource descriptions. Resource descriptions are used to associate resource groups with CICS systems or system groups.

To define resource groups from the main menu (Figure 5-4 on page 110), select **Administration views** → **Fully functional BAS administrative views** → **Resource groups** → **Create**.

Specify the resource group name. Leave Model value as No and click **Yes** to create a resource group, as shown in Figure 5-16 on page 122.

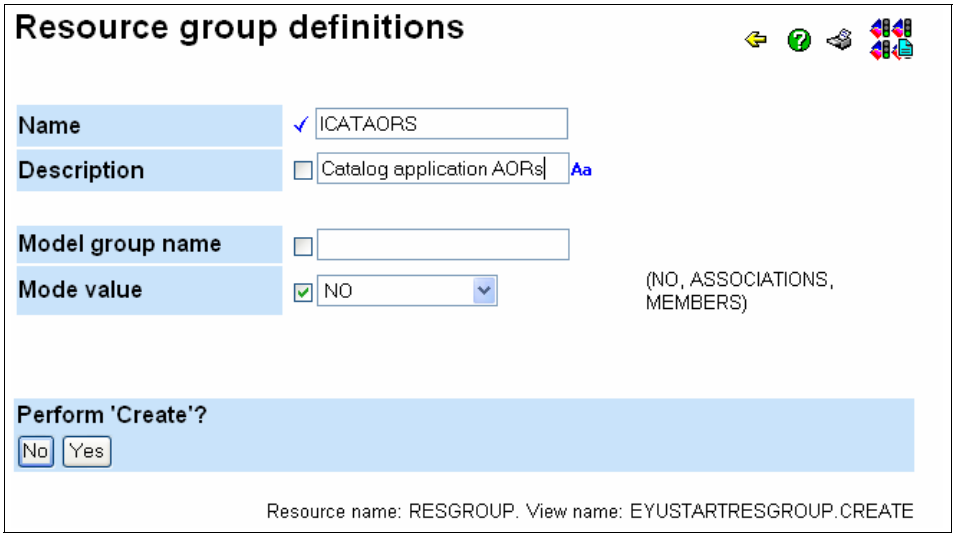


Figure 5-16 Resource group definition

Create the resource groups ICATTORS, SCSCPTA1, SCSCPTA2, SCSCPTAA1, and SCSCPAA4 following the same sequence of steps.

Setting up resource descriptions

Resource descriptions are used for associating resource groups to CICS systems or system groups.

To define resource descriptions from the main menu (Figure 5-4 on page 110), select **Administration views** → **Fully functional BAS administrative views** → **Resource descriptions** → **Create**.

Specify the Resource description name, as shown in Figure 5-17 on page 123. The Resource group scope name is the name of the CICS system or system group where this description will be installed. The Autoinstall request type must be Yes in order to have CICSPlex SM install the definitions at CICS startup.

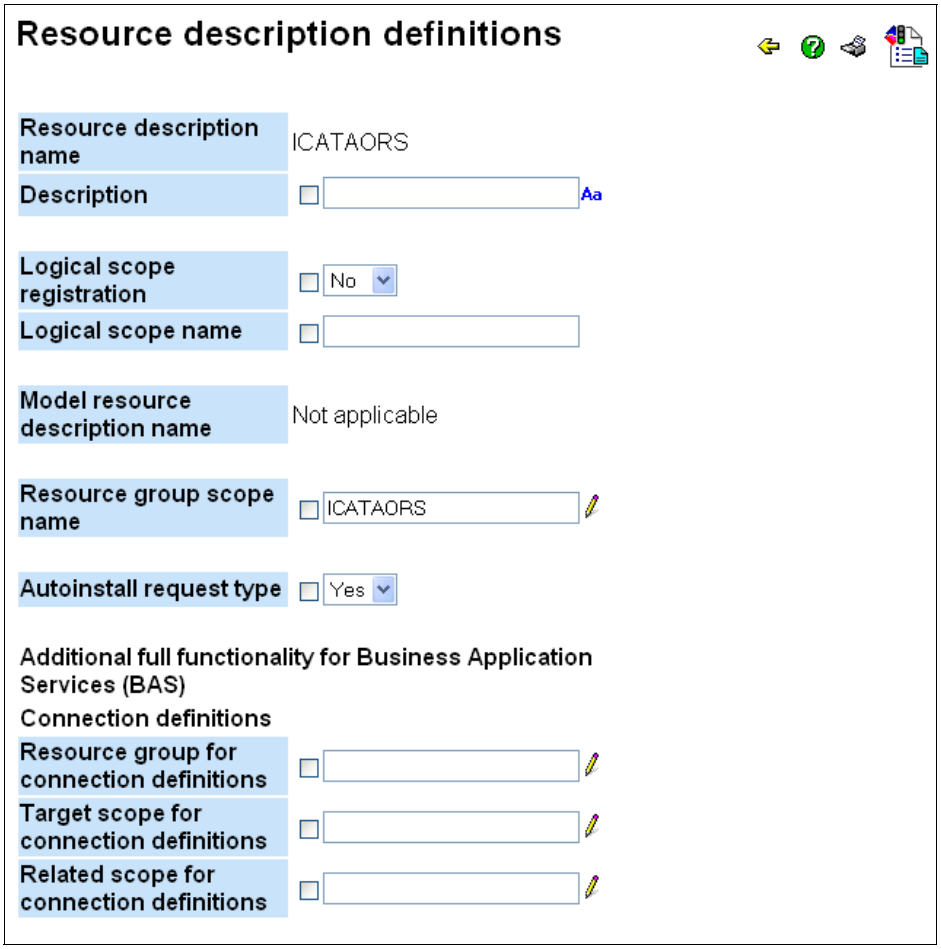


Figure 5-17 Resource description definition

Now create resource descriptions for ICATTORS, SCSCPTA1, SCSCPTA2, SCSCPAA1, and SCSCPAA4 following the same sequence of steps.

Connecting resource groups to resource descriptions

To connect resource groups to resource descriptions from the main menu (Figure 5-4 on page 110), select **Administration views** → **Fully functional BAS administrative views** → **Resource group**.

Select the resource group to add and click **Add to Resource description**, as shown in Figure 5-18.

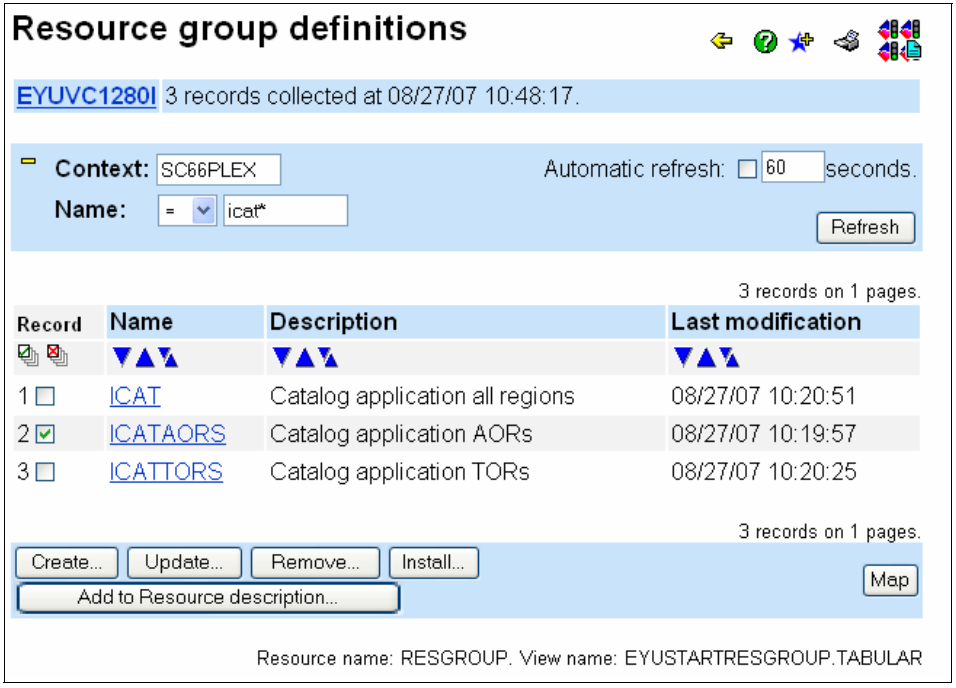


Figure 5-18 Resource group definition

Fill in the resource description the resource group is to be added to and click **Yes**, as shown in Figure 5-19 on page 125.

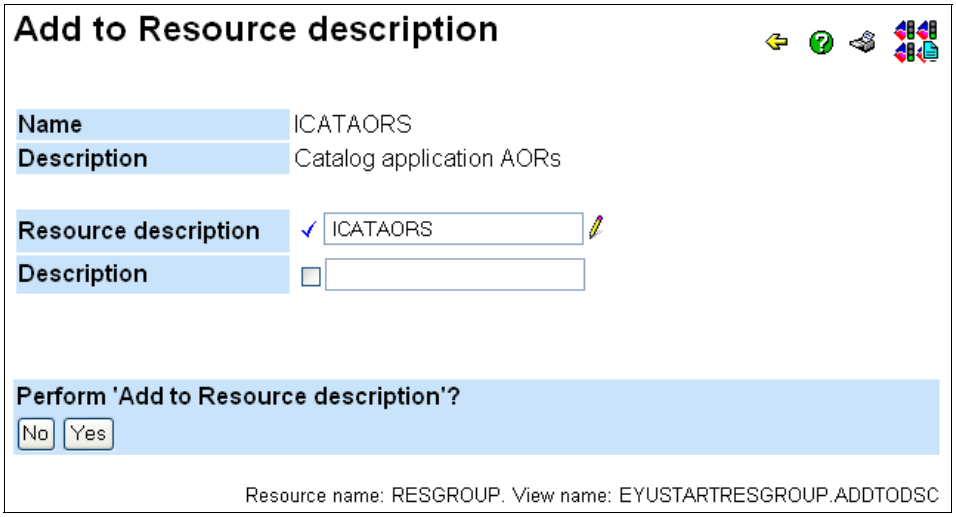


Figure 5-19 Add to Resource description

Now add resource groups ICATTORS, SCSCPTA1, SCSCPTA2, SCSCPAA1, and SCSCPAA4 to the respective resource descriptions following the same directions.

5.5.4 CICS interregion resource definitions

In this section, we describe the CICS system definitions that have to be made for running the application.

Interregion connections

IP Interconnectivity Connections is a new type of intercommunication link that CICS TS V3.2 can use. We use IP connections to connect the CICS regions. In order to use IP connections, each region must have a TCP/IP service for this purpose. Furthermore, each link has to be defined to CICS. The TCP/IP service is used for inbound traffic and the IP connections are used for outbound traffic. The same TCP/IP service can be used for all IP connection within a CICS region.

The same IP Connection definitions from the front-end regions to a given back-end region can be used in all front-end regions. The TCP/IP service definitions have to be unique in all regions since they specify the TCP/IP address and port number.

TCP/IP service definitions

To define the TCP/IP services from the main menu (Figure 5-4 on page 110), select **Administration views** → **Fully functional BAS administrative views** → **Resource definitions** → **TCP/IPdefinitions** → **Create**.

Fill in the various parameters and click **Yes** to create a new TCP/IP service definition, as shown in Figure 5-20 and Figure 5-21 on page 127.

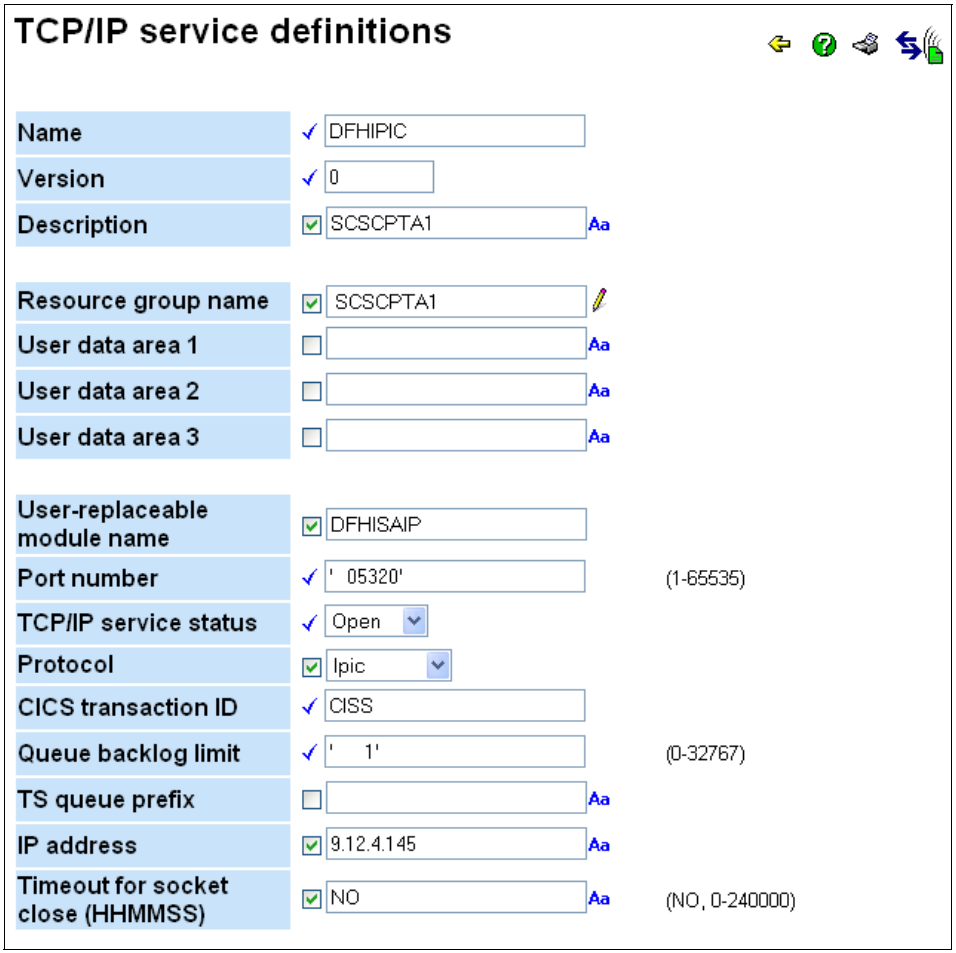


Figure 5-20 TCP/IP service view Part 1

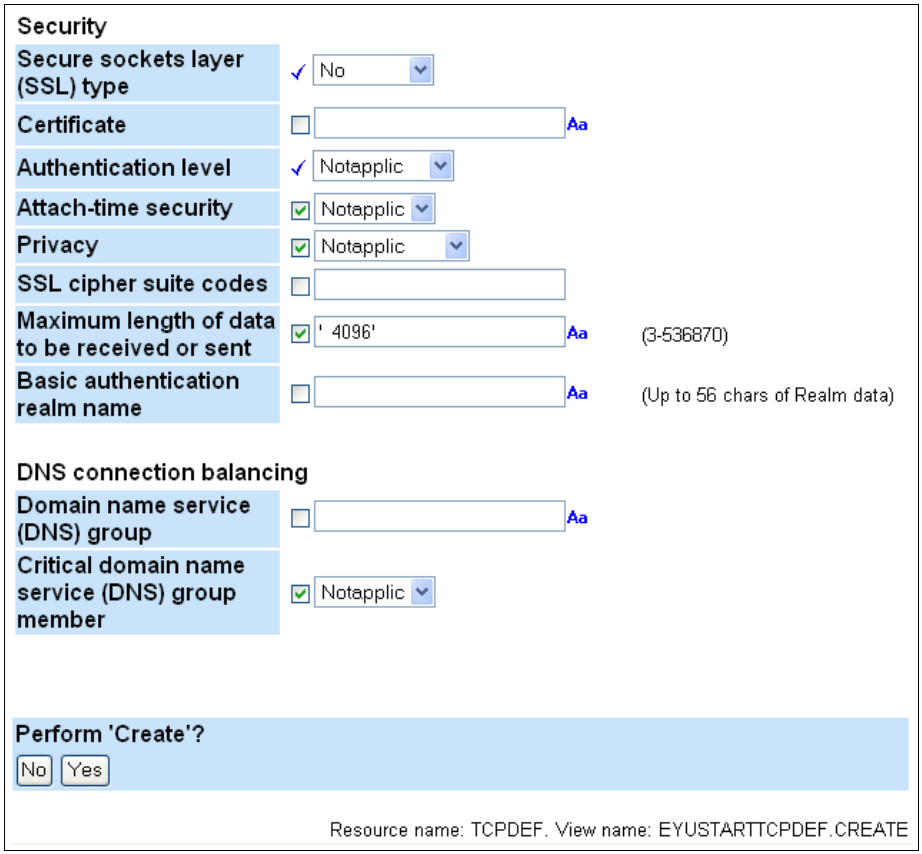


Figure 5-21 TCP/IP service view Part 2

Note: The user-replaceable module name must be DFHISAIP for IP connection TCP/IP services. The protocol must specify Ipic.

Follow the same directions for defining TCP/IP services for SCSCPTA2, SCSCPAA1, and SCSCPAA4.

Defining IP connections

To define the IP connections from the main menu (Figure 5-4 on page 110), select **Administration views > Fully functional BAS administrative views > Resource definitions > IPIC connection definitions >Create**

Complete the required fields shown in Figure 5-22 and scroll to the next window (Figure 5-23 on page 129).

Figure 5-22 New IP connection definition - part 1

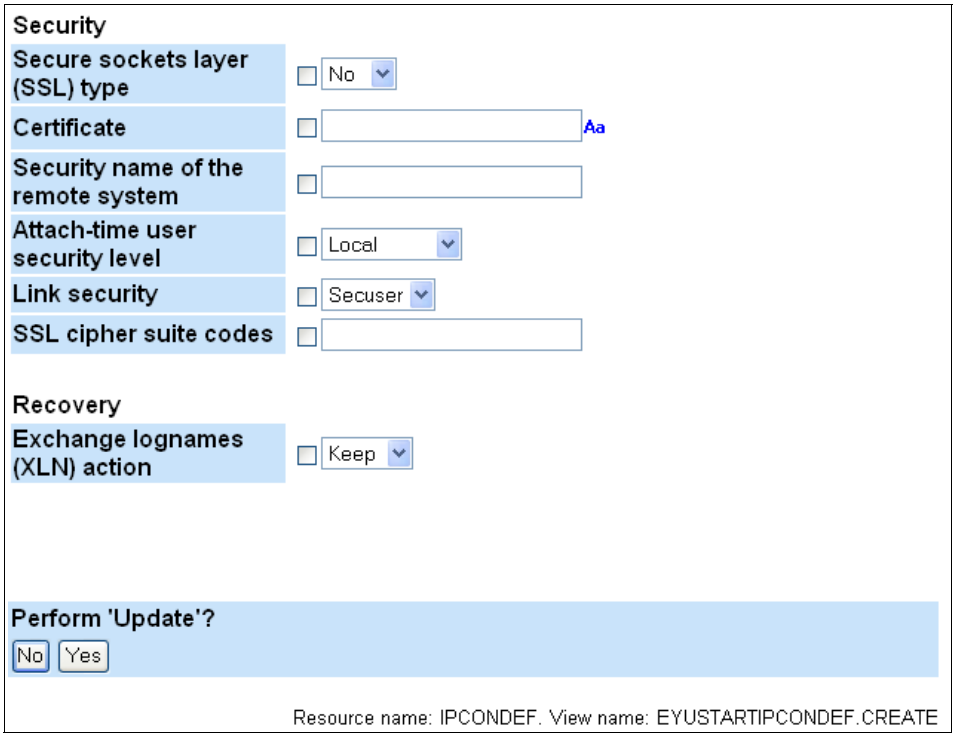


Figure 5-23 New IP connection definition - page 2

Complete the required fields and click **Yes** to create the definition.

IP Connections for SCSCPTA1, SCSCPTA2, and SCSCPAA4 should be created in a similar way.

Note: The name of the IP connection is the SYSID by which CICS knows the remote system.

5.5.5 CICS application resource definitions

Here we discuss the CICS application resource definitions.

VSAM file definition

To define the VSAM file from the main menu (Figure 5-4 on page 110), select **Administration views** → **Fully functional BAS administrative views** → **Resource definitions** → **File definitions** → **Create**.

Complete the required fields shown in Figure 5-24 on page 131 and scroll to the next page (Figure 5-25 on page 132).

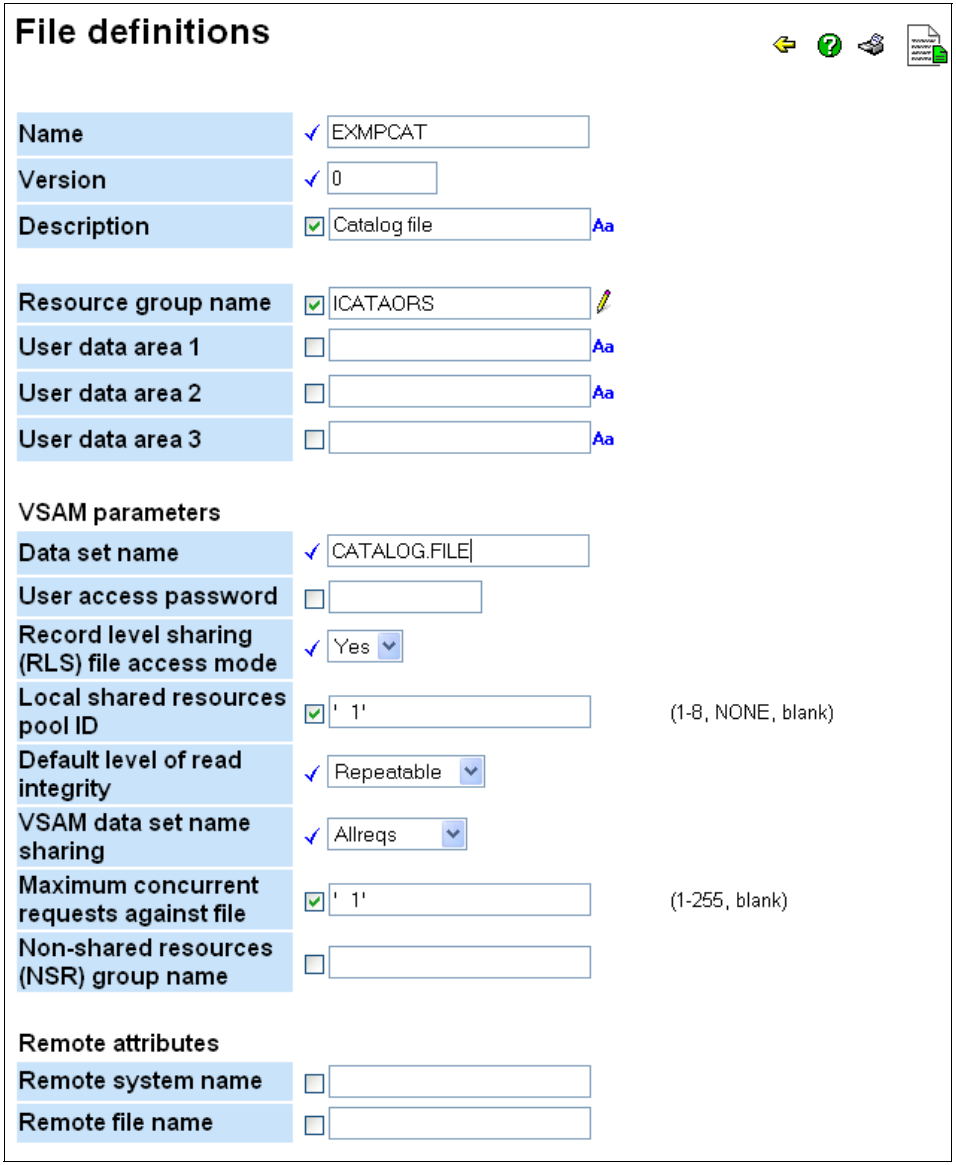


Figure 5-24 New file definition, part 1

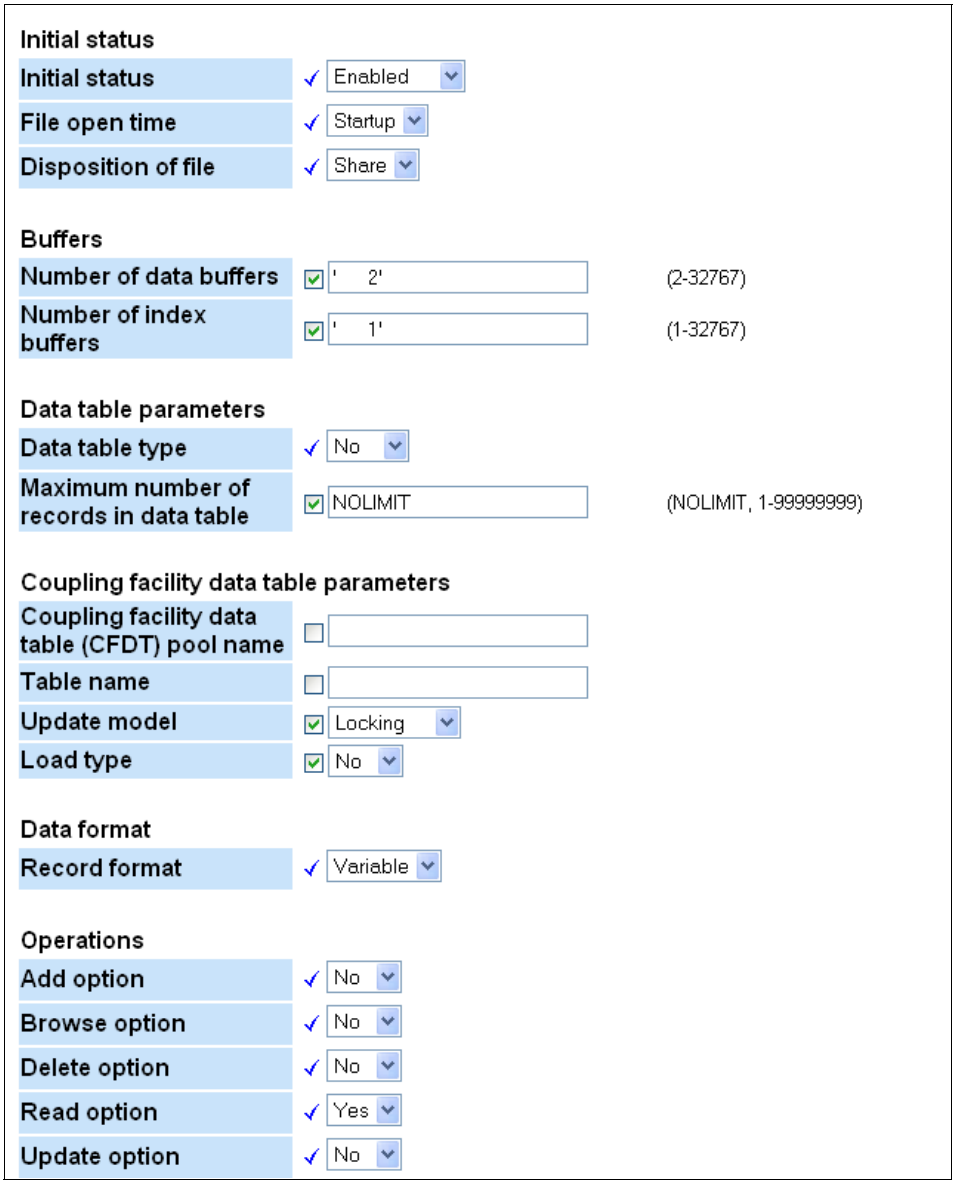


Figure 5-25 New file definition, page 2

Complete the required fields and scroll to the next page (Figure 5-26).

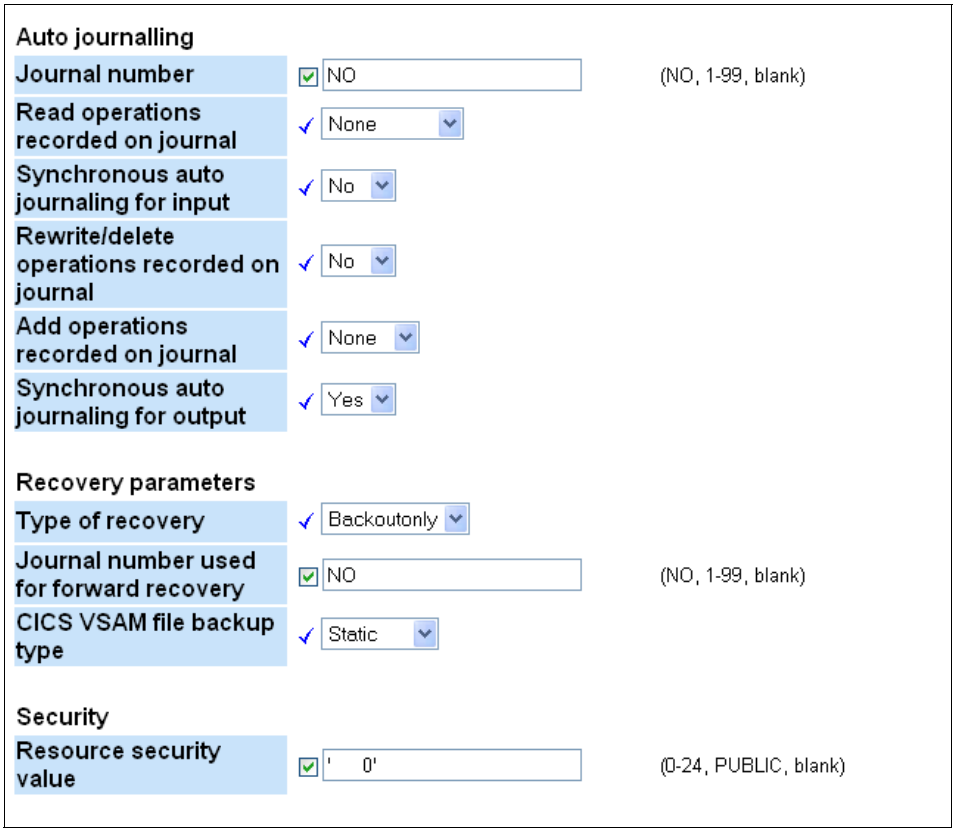


Figure 5-26 New file definition Part 3

Complete the required fields and scroll to the next page (Figure 5-27).

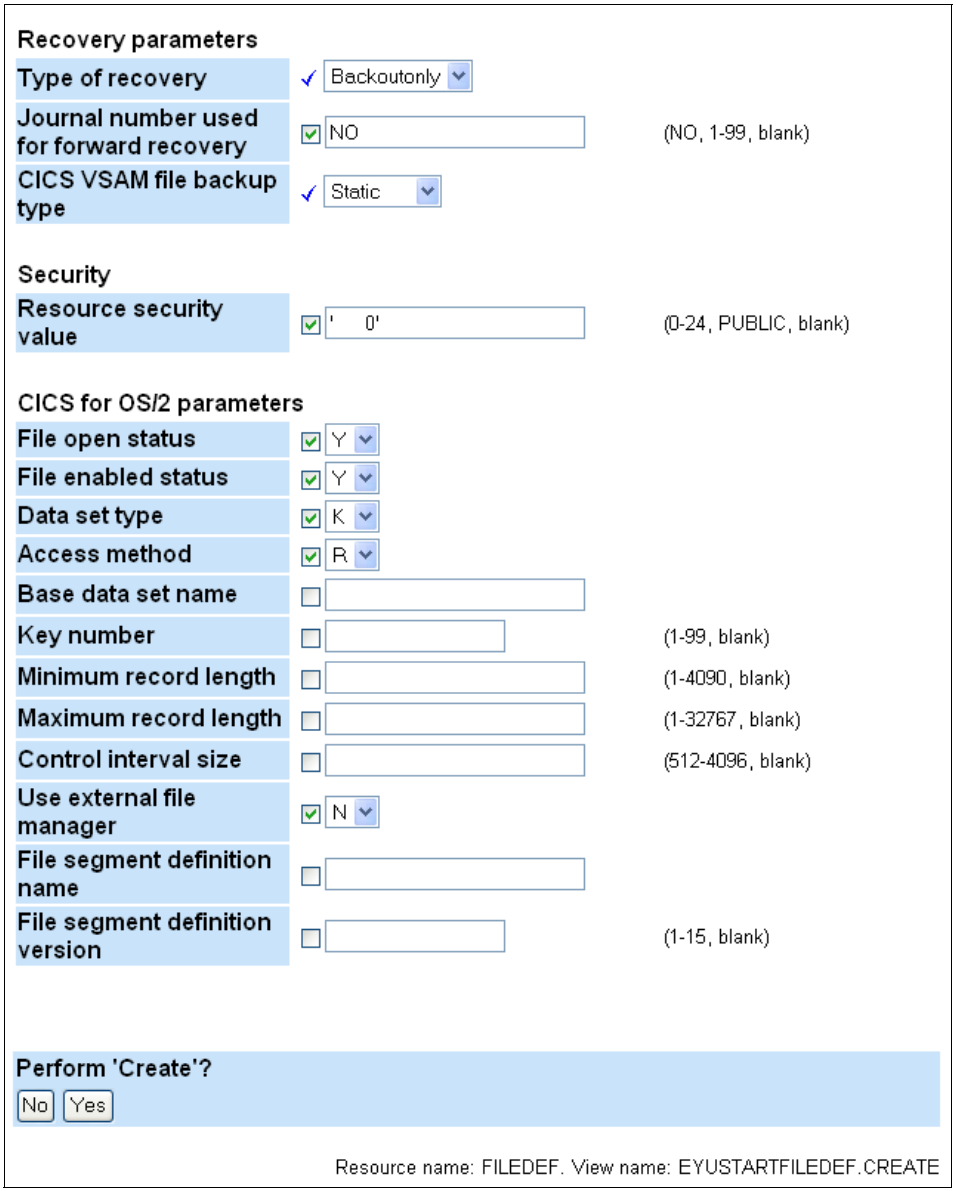


Figure 5-27 New file definition Part 4

Complete the required fields and click **Yes** to create the session definition.

Program definition

To define the program DFH0XCMN from the main menu (Figure 5-4 on page 110), select **Administration views** → **Fully functional BAS administrative views,** → **Resource definitions** → **Program definitions** → **Create**. Fill in all the parameters, as shown in Figure 5-28 and Figure 5-29 on page 136.

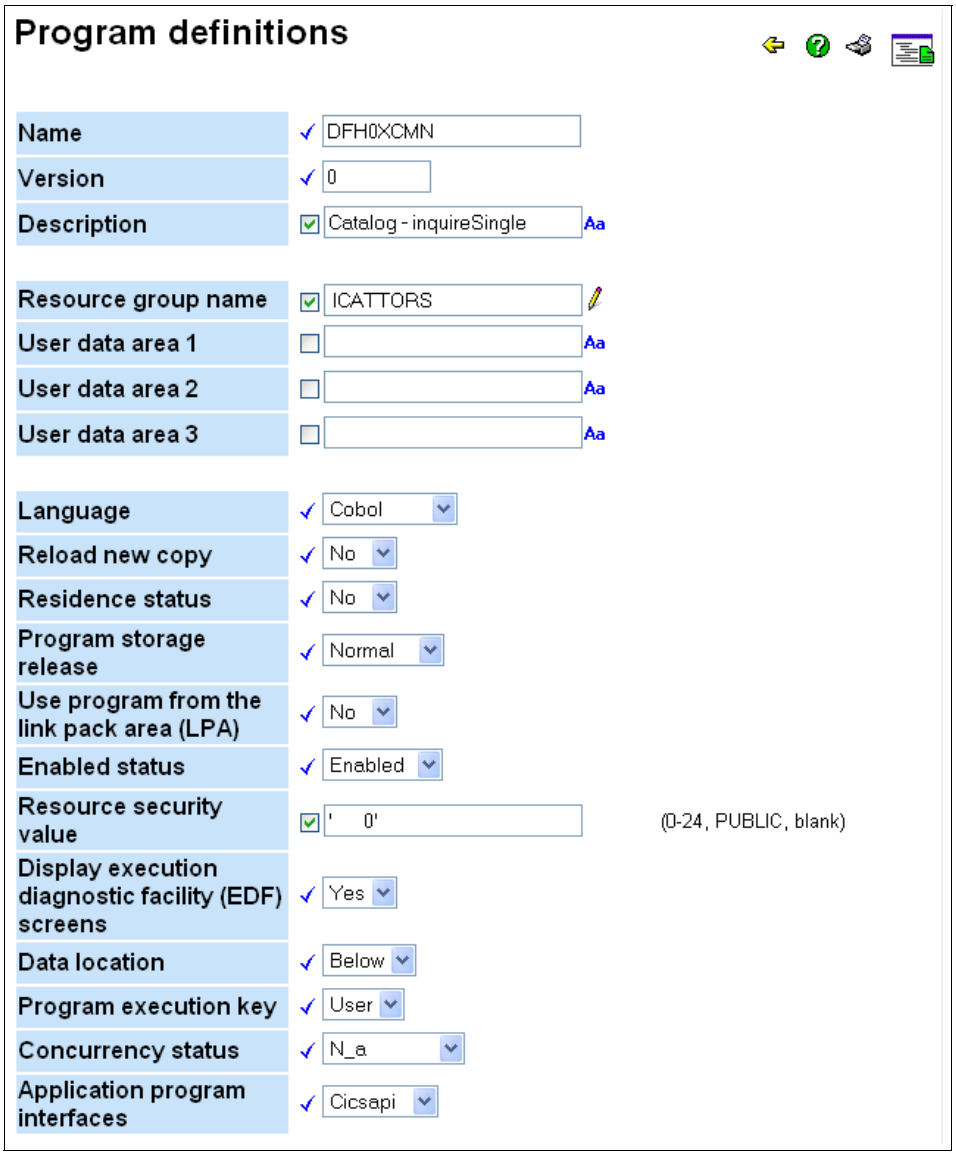


Figure 5-28 Program definition Part 1

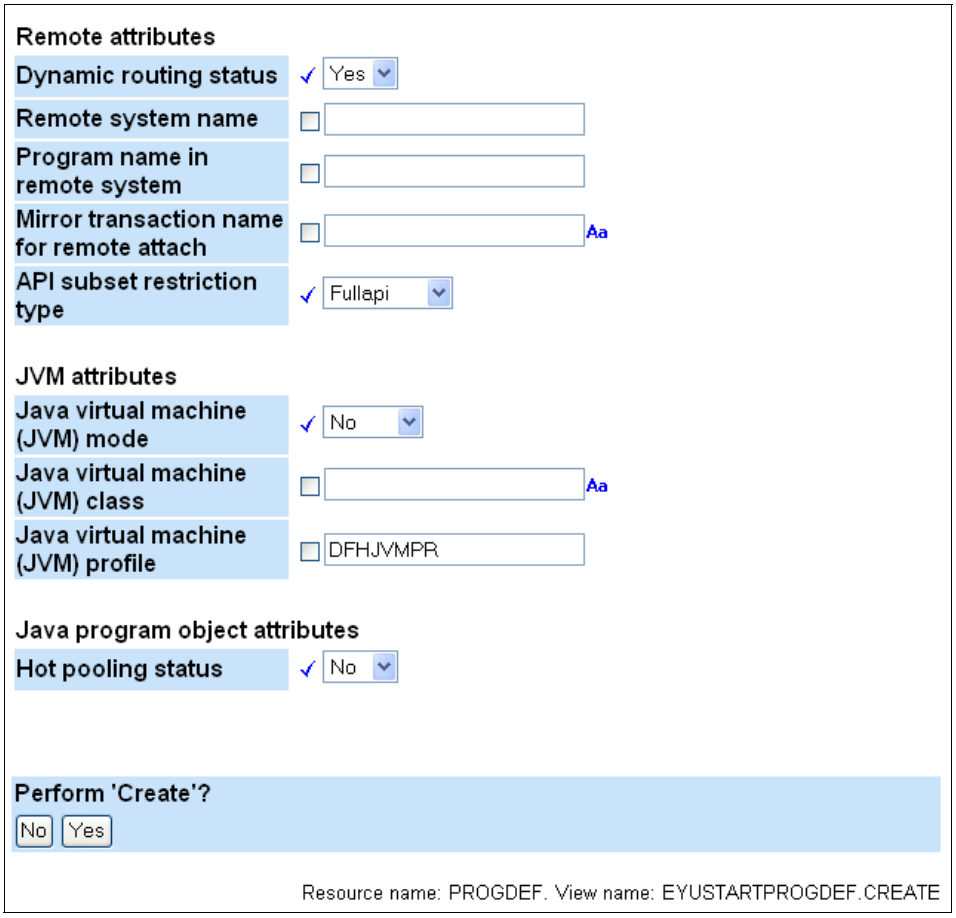


Figure 5-29 Program definition Part 2

After filling in all relevant parameters, click **Yes** to create definition.

Note: The program has to be defined as dynamic in order to enable dynamic DPL.

Pipeline definition

To define the pipeline used for the Web service from the main menu (Figure 5-4 on page 110), select **Administration views** → **Fully functional BAS administrative views** → **Resource definitions** → **Pipeline definitions** → **Create**.

Enter all the fields and click **Yes** to create the pipeline link definition, as show in Figure 5-30.

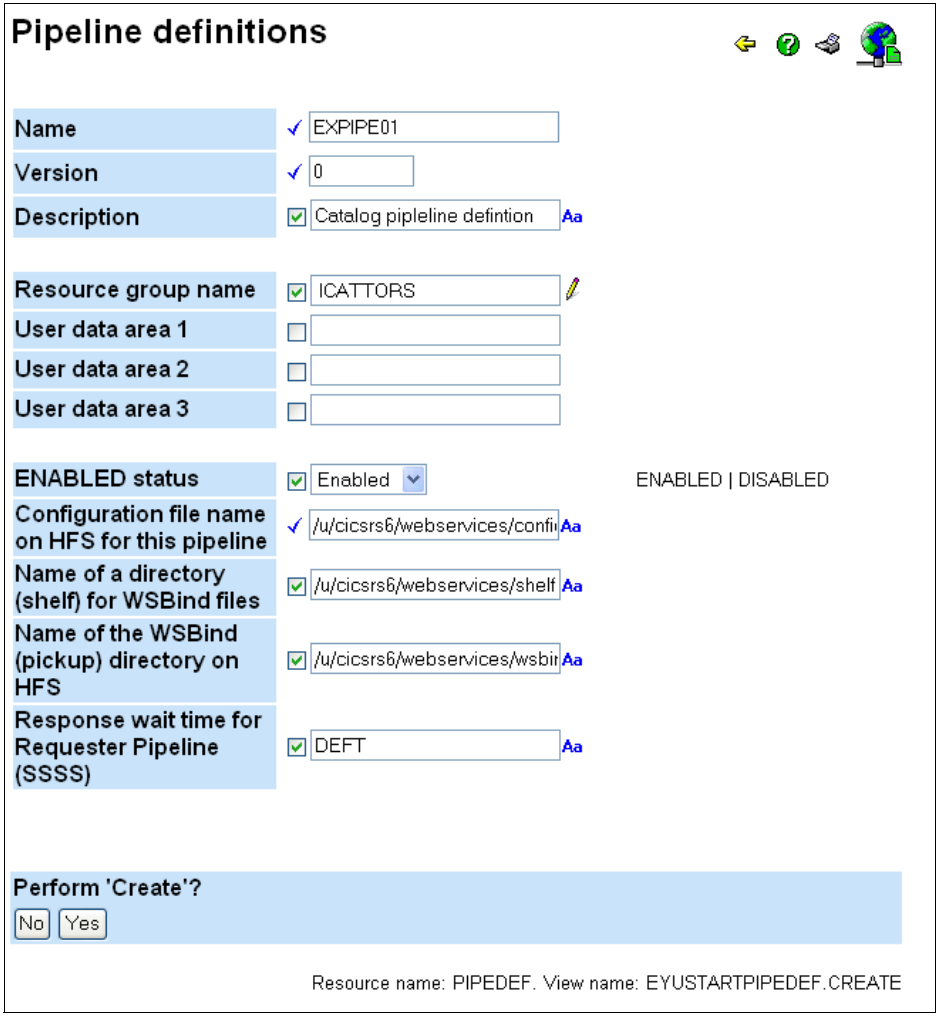


Figure 5-30 Pipeline definition

The configuration file name is the name of the HFS file, where your Web service configuration file is found.

Note: If you use the CICS-provided pipeline configuration file basicsoap11provider.xml, do not forget to change the encoding parameter specified on the first line of the file if your Web service requester is a WebSphere client. It has to be changed from encoding="EBCDIC-CP-US"?> to encoding="UTF-8"?>.

URI map definition

To define the URI maps used for the Web services from the main menu (Figure 5-4 on page 110), select **Administration views** → **Fully functional BAS administrative views** → **Resource definitions** → **URI mapping definitions** → **Create**.

Enter the required parameters, as shown in Figure 5-31, and scroll down (Figure 5-32 on page 140).

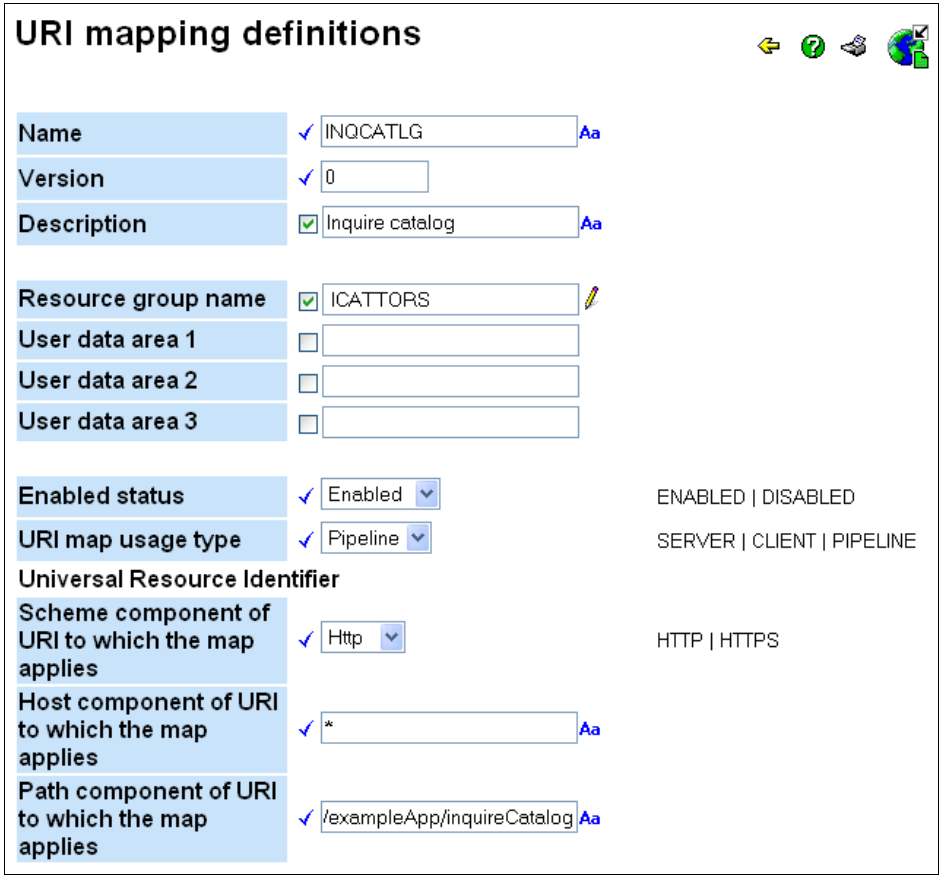


Figure 5-31 URI mapping definition part 1

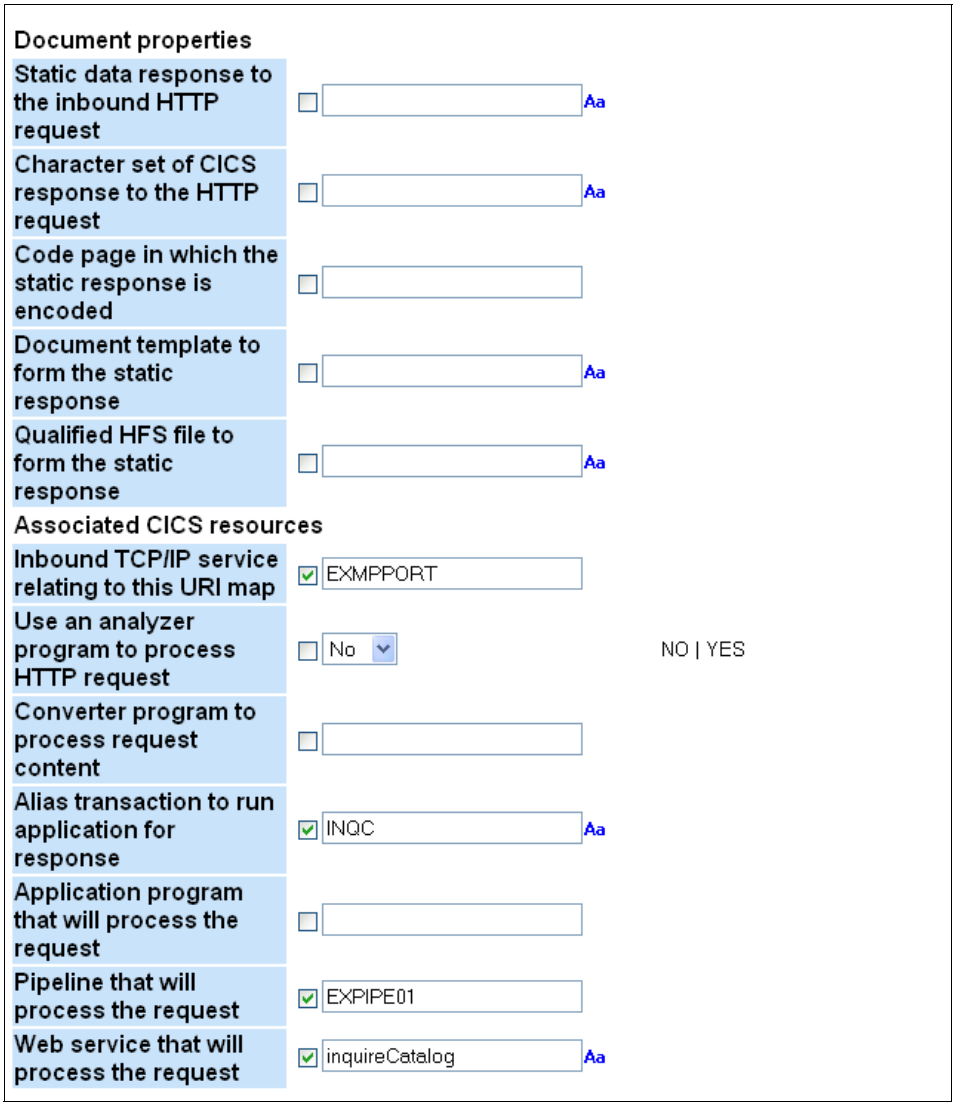


Figure 5-32 URI mapping definition part 2

Fill in TCP/IP service, pipeline, and Web service. Then scroll to the bottom and click **Yes** to create the definition.

URI mappings should be defined for InquireSingle and placeOrder too.

Web service definition

To define the Web service used for the Web services from the main menu (Figure 5-4 on page 110), select **Administration views** → **Fully functional BAS administrative views** → **Resource definitions** → **Web services definitions** → **Create**.

Fill in the required parameters and click **Yes** to create the definition, as shown in Figure 5-33.

Figure 5-33 Web service definition

Web services should be defined for InquireSingle and placeOrder too.

Transaction definitions for the TORs

To define the transactions to the TORs from the main menu (Figure 5-4 on page 110), select **Administration views** → **Fully functional BAS administrative views** → **Resource definitions** → **Transaction definitions** → **Create**. The window in Figure 5-34 should appear.

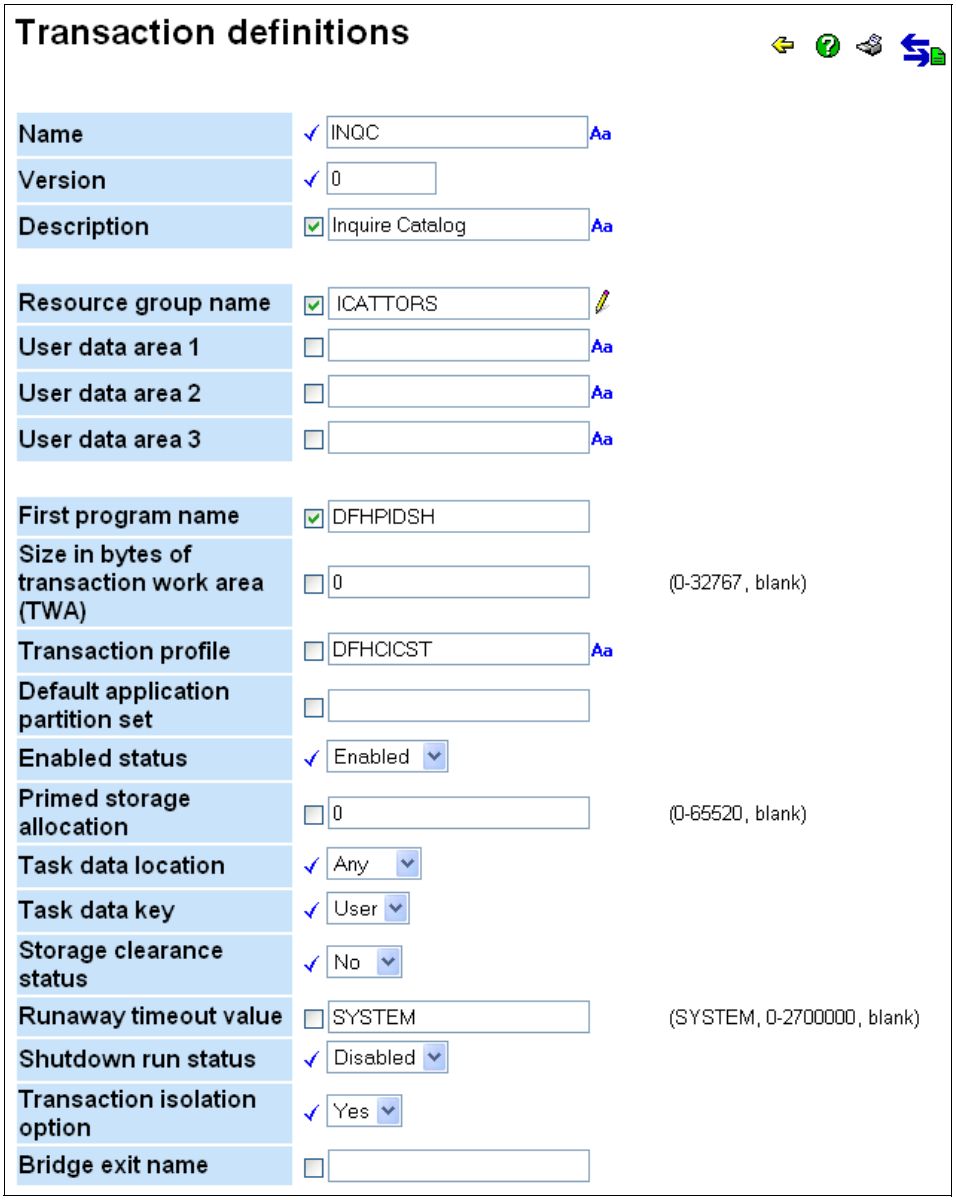


Figure 5-34 Transaction definition to the TORs

Fill in the required parameters, scroll down, and click **Yes** to create. Transactions INQS and UPDT should be defined in the same way.

Note: The first program must be DFHPIDSH.

Transaction definitions for the AORs

To define the transactions to the AORs from the main menu (Figure 5-4 on page 110), select **Administration views** → **Fully functional BAS administrative views** → **Resource definitions** → **Transaction definitions** → **Create**.

Fill in the required parameters and scroll down and click **Yes** to create the definitions, as shown in Figure 5-35. Transactions INQS and UPDT should be defined in the same way.

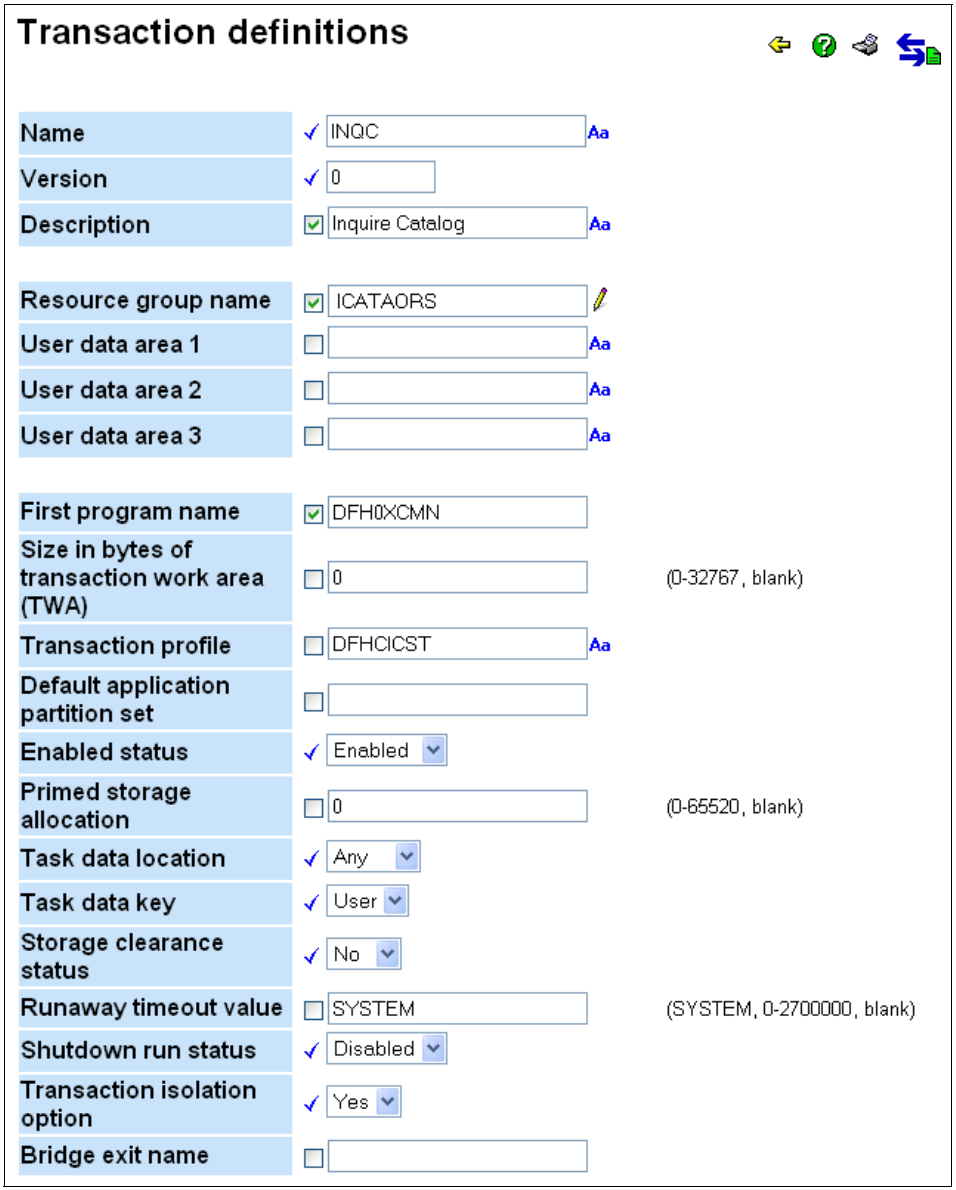


Figure 5-35 Transaction definition to the AORs

Note: The first program must be DFH0XCMN.

5.5.6 The MAP function

The CICSPlex SM map function provides a map of the business application services definitions. The resource where the function is selected is the starting point for the resources displayed.

With CICS TS V3.2, the map function has been implemented in the Web User Interface.

Mapping resource descriptors

To map a resource description from the main menu (Figure 5-4 on page 110), select **Administration views** → **Fully functional BAS administrative views** → **Resource descriptions**. The window shown in Figure 5-36 should display.

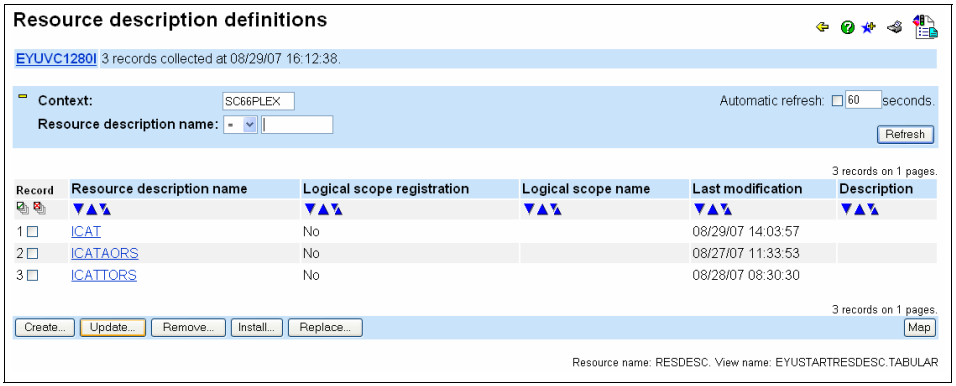


Figure 5-36 List of resource descriptions

Select resource description **ICATTORS** and click **Map**. The window shown in Figure 5-37 should display.

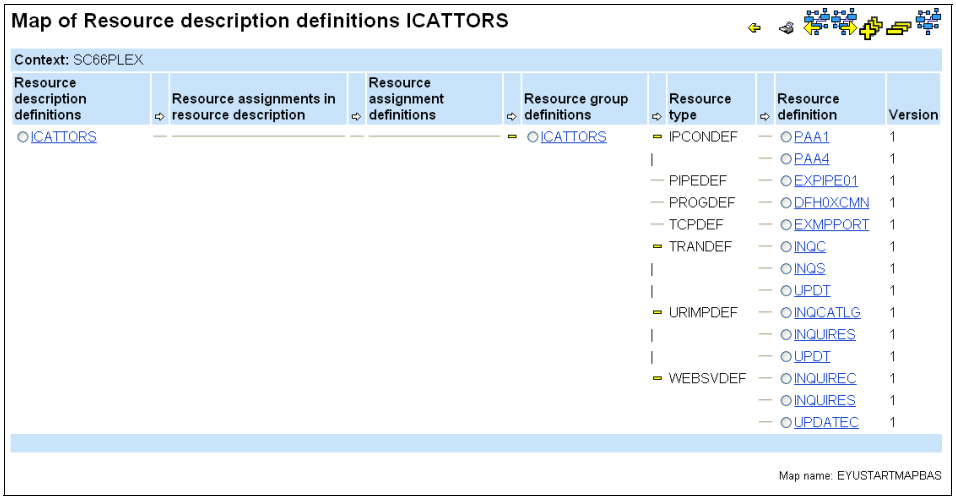


Figure 5-37 Map of resource description ICATTORS

Mapping CICS resources

To map a resource definition, in this case an IP connection, from the main menu (Figure 5-4 on page 110), select **Administration views** → **Fully functional BAS administrative views** → **Resource definitions** → **IPIC connection definitions**. The window shown in Figure 5-38 should be displayed.

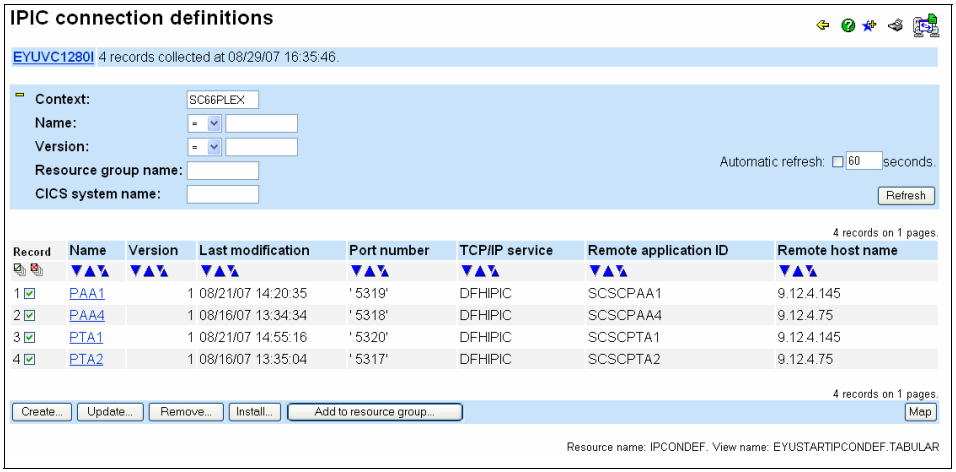


Figure 5-38 Selecting definitions

Select the definitions to map and click **Map**. The window shown in Figure 5-39 should be displayed.

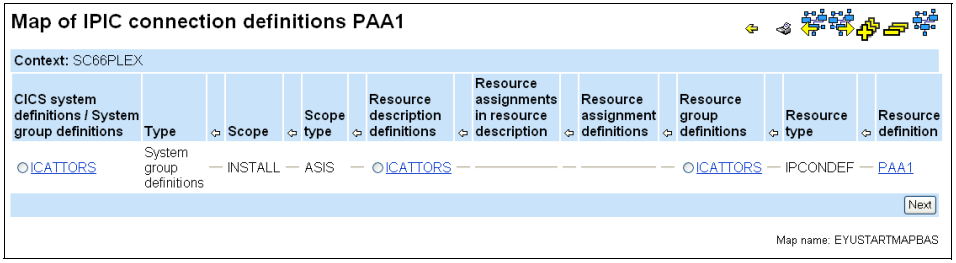


Figure 5-39 Map of IP connection PAA1

If more than one resource is selected, you may click **Next** to display the map.

5.6 Workload management

You have to set up CICSPlex SM workload definitions to be able to run the Web services pipeline in the front-end region and invoke the business application program in the back-end using a dynamic program link.

The following definitions have to be set up:

1. A workload management specifications has to be defined.
2. A workload management group has to be defined and added to the specification.
3. A workload management definition needs to be defined and added to the workload management group.
4. The workload management specification has to be associated with a CICS system or CICS system group.

Tip: The scope of the target system can be either a single CICS region or a system group. Normally a system group would be used, because doing dynamic routing to a single target has no meaning. Also, using a system group enables further AORs to be added to the routing definition without having to reinstall the workload definition into the active workload.

All of the workload management definition views can be located in the Workload manager administration views (Figure 5-8 on page 114).

5.6.1 Workload management specification

The WLM specification (WLMSPEC) is used for associating routing regions with the default target regions.

Notes:

- A CICS target region can be a member of multiple WLMSPECs.
- A CICS front-end routing region can only be an initiator region in one WLMSPEC.
- When you use the SIT option DSRTPGM, the back-end region must also be added as a routing region, so in this circumstance it can only be in one workload management specification.

To define a WLM specification from the main menu (Figure 5-4 on page 110), select **Administration views** → **Workload manager administration views** → **Specifications** → **Create**. The window shown in Figure 5-40 should be displayed.

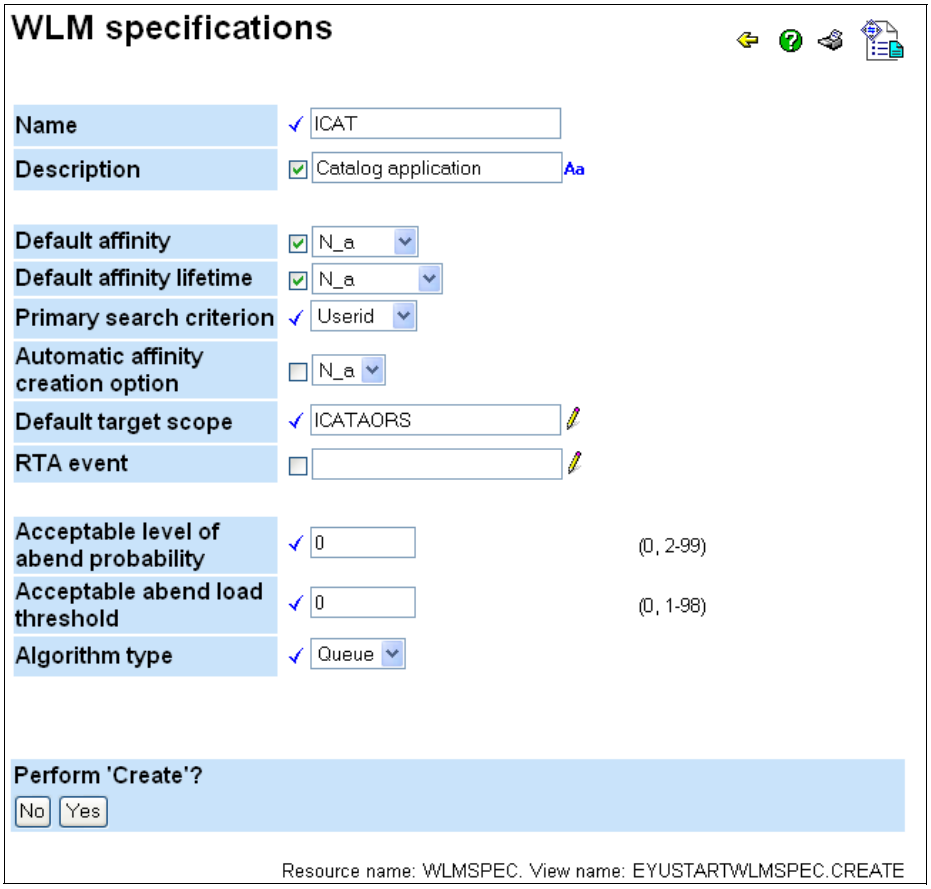


Figure 5-40 WLM specification definition

Fill in the parameters and click **Create**. Then select the WLM specification **ICAT** and click **Associate CICS group**. The window shown in Figure 5-41 should be displayed.

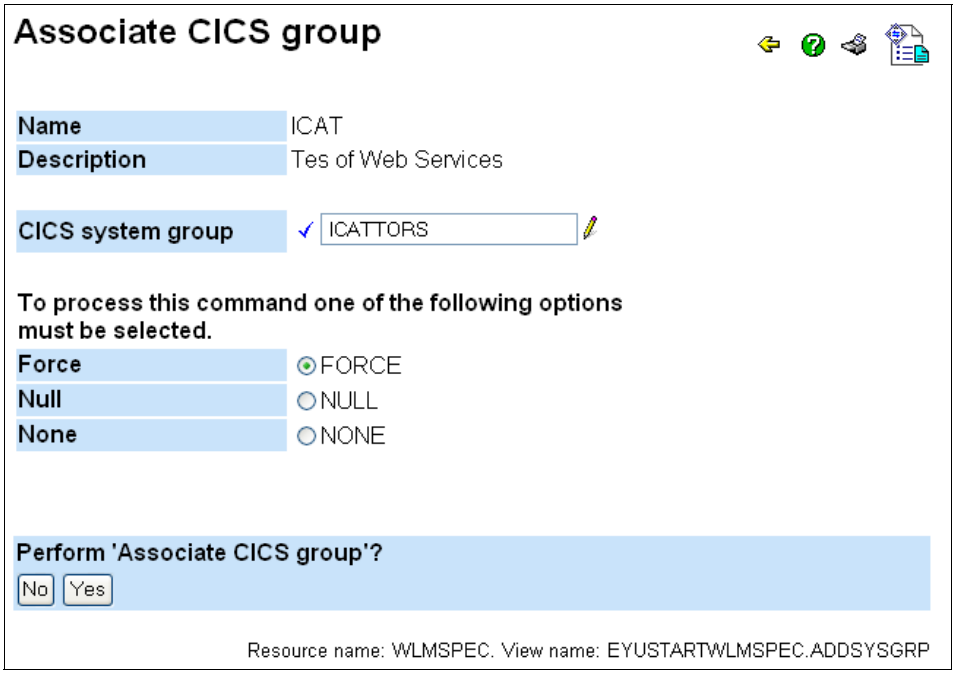


Figure 5-41 Associate CICS system group to WLM specification

Enter the CICS system group and select **Force**. Click **Yes** to update.

Workload management group

To define a WLM group from the main menu (Figure 5-4 on page 110), select **Administration views > Workload manager administration views > Groups > Create**. The window shown in Figure 5-42 on page 151 should be displayed.

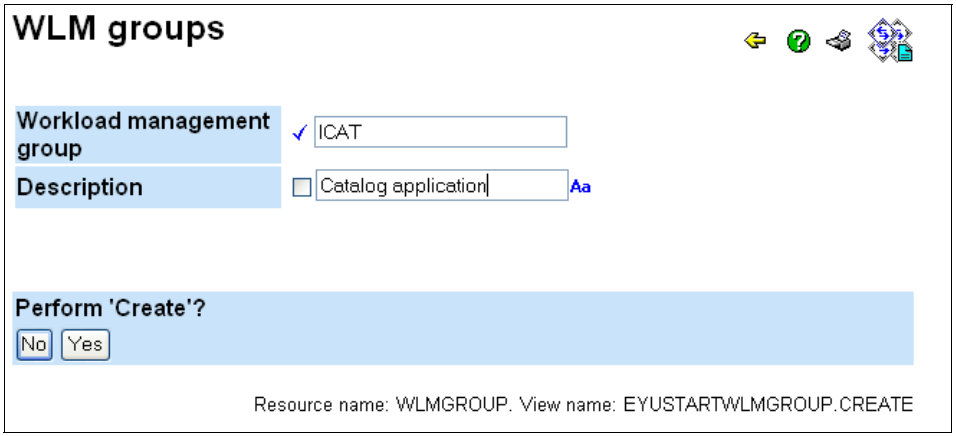


Figure 5-42 Workload management group definition

Fill in the parameters and click **Create**. Then select WLM group **ICAT** and click **Add to WLM specification**, as shown in Figure 5-43.

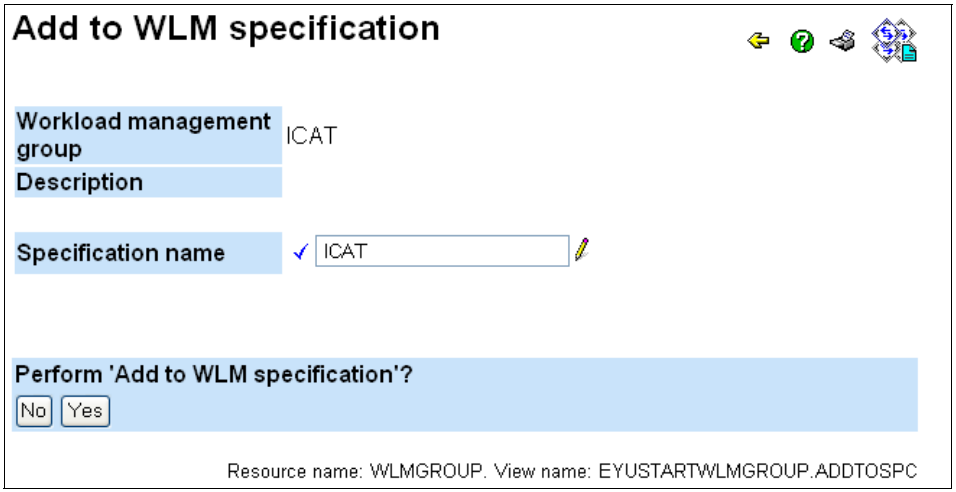


Figure 5-43 Adding WLM group to a WLM specification

Enter the WLM specification name and click **Yes** to add the WLM group.

Workload management definition

To define a WLM definition from the main menu (Figure 5-4 on page 110), select **Administration views** → **Workload manager administration views** → **Definitions** → **Create**. The window shown in Figure 5-44 is displayed.

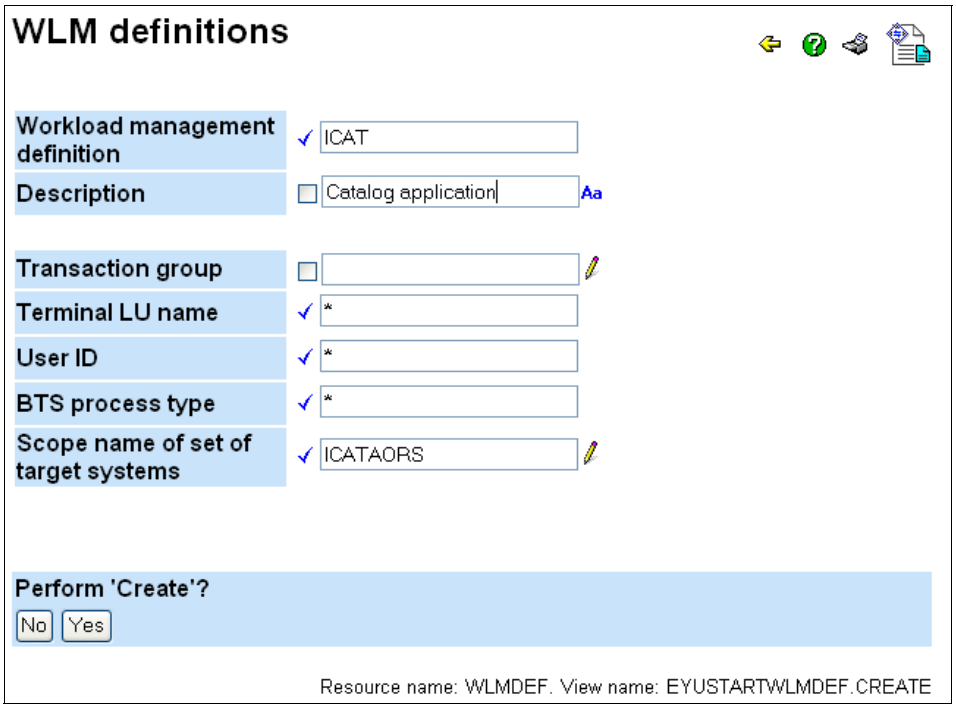


Figure 5-44 Defining a WLM group

Enter the WLM definition name. Terminal LU name and User ID must be asterisk. Then click **Yes** to create. In the window shown in Figure 5-45 on page 153, select WLM definition **ICAT** and click **Add to WLM group**.

Note: Since we are only using a dynamic program link, we do not need the Transaction group.

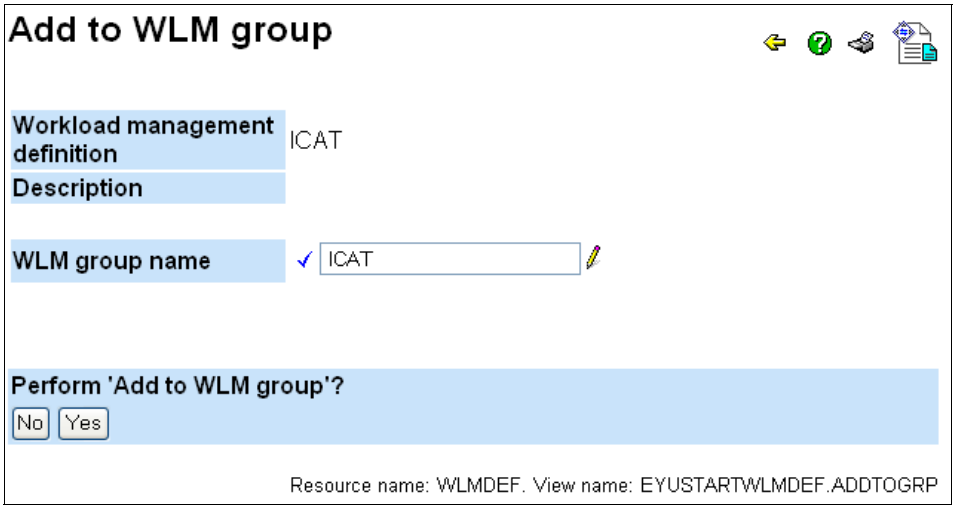


Figure 5-45 Adding a WLM definition to a WLM group

Enter the WLM group name and click **Yes** to update.

Mapping of WLM definitions

Using the map function, we can get a clear picture of the WLM definitions.

From the main menu (Figure 5-4 on page 110), select **Administration views** → **Workload manager administration views** → **Specifications**. Then select the specification and click **MAP**. The window shown in Figure 5-46 is displayed.

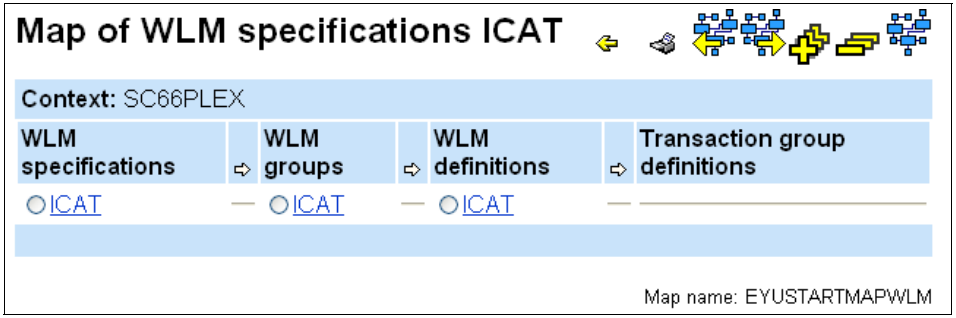


Figure 5-46 Map of the WLM specification ICAT

5.7 Managing the environment

Managing the CICS and application infrastructure in an on demand environment creates new challenges in providing access to high availability systems while at the same time being able to make changes to the infrastructure and applications.

The CICSPlex SM Application Programming Interface (API) can be used to migrate definitions, install new definitions, and refresh programs as part of the change control process.

Using the CICSPlex SM API, REXX™ routines can be written to run in batch as part of the change management process. Sample routines have been written to perform the following functions:

- Install a resource definition.
- Copy a resource definition from one CICSplex to another.
- Phase in a new copy of a CICS program.
- Phase out a JVM™.
- Perform a pipeline scan.

5.7.1 Migrating definitions

Migrating definitions between two different CICSplexes can be time consuming and cumbersome using the CICSPlex SM BATCHREP utility.

To provide assistance to those that are running multiple CICSplexes, we have written a sample routine to copy CICS resource definitions from one CICSplex to another. The sample routine has been written to copy a transaction or file definition between two CICSplexes that share the same CMAS. The REXX program can easily be enhanced to include other CICS resource types.

5.7.2 Installing definitions

Installing a new or modified BAS definition can always be done by performing a CICS cold start. In some circumstances, it may not be possible to shut down the CICS regions so that a cold start can be performed. In these cases, a simple BAS install can be done to install all the resources into the required CICS regions without having to log on to each CICS region.

When installing a CICS resource, the easiest way to install the resource is to make use of the resource assignment. This way there is no need to worry about which region to install the resource into, or any overrides. Using the resource assignment to install the definition takes care of these options. However, in our configuration, we have no remote definitions and therefore are not using resource assignments.

A sample REXX program has been written to perform the installations, or the WUI can be used by going to the resource group display for that resource. See Appendix A, “Sample REXX scripts” on page 229 for more information.

To display the CICS resource group definitions from the main menu (Figure 5-4 on page 110), select **Administration views** → **Fully functional BAS administration views** → **Resource groups**. You should see the window shown in Figure 5-47.

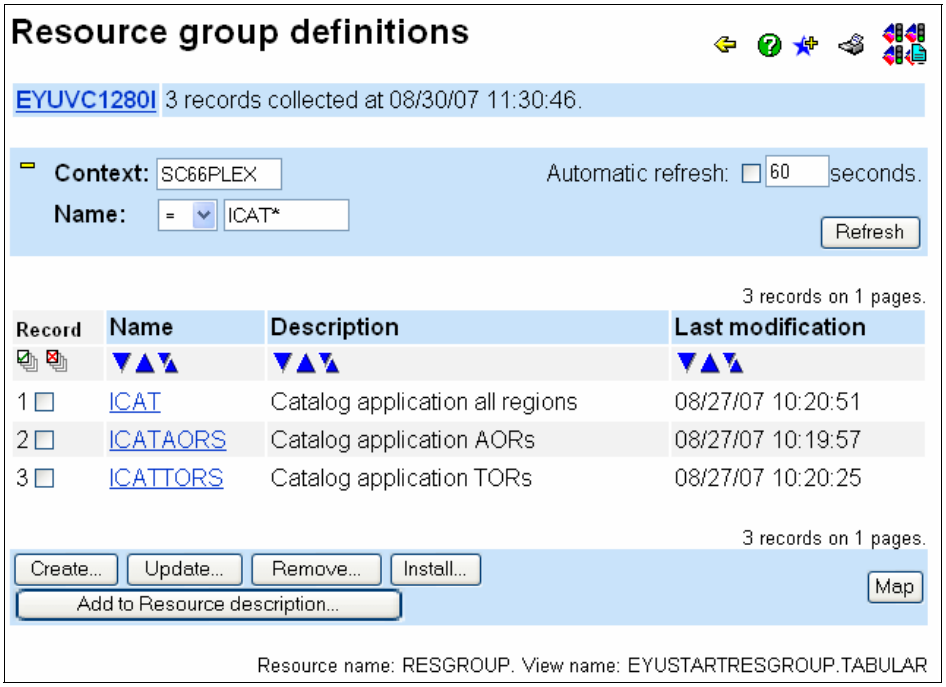


Figure 5-47 Resource group definitions view

Select the group the resource is in and click **Install** (Figure 5-48).

Figure 5-48 Installing a CICS resource

In the target scope value, enter the CICS system name or CICS system group name and select the type of resource to install.

Click **Yes** to install the resource definition.

5.7.3 Refreshing resources

CICS resources are commonly updated when a new version of resource definitions is required by the application. CICS does not need to be shut down for these resource definitions to be refreshed. Making use of CICSPlex SM allows you to refresh multiple resources in multiple regions on multiple LPARS at the same time.

Sample routines have been written to allow you to refresh some of these resources and can be easily be added to an automated change management system.

The following scenario demonstrates how easy and simple it is to refresh a program through the WUI. This same scenario can be applied to other resources such as JVMs and pipelines.

When CICS programs are updated, CICS needs to reload the program before the new copy of the code is used by CICS. This is normally done by using the CEMT S PROGRAM(xxxx) PHA.

To display the CICS program view from the main menu (Figure 5-4 on page 110), select **CICS operation views** → **Program operations views** → **Programs**. You should see the window shown in Figure 5-49.

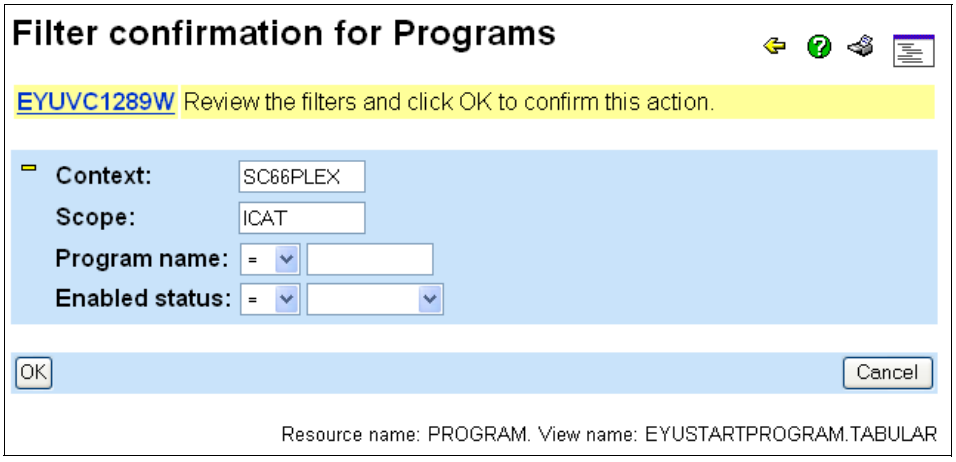


Figure 5-49 Program view

Ignore the error message and enter the program name and click **Refresh** to display the programs (Figure 5-50).

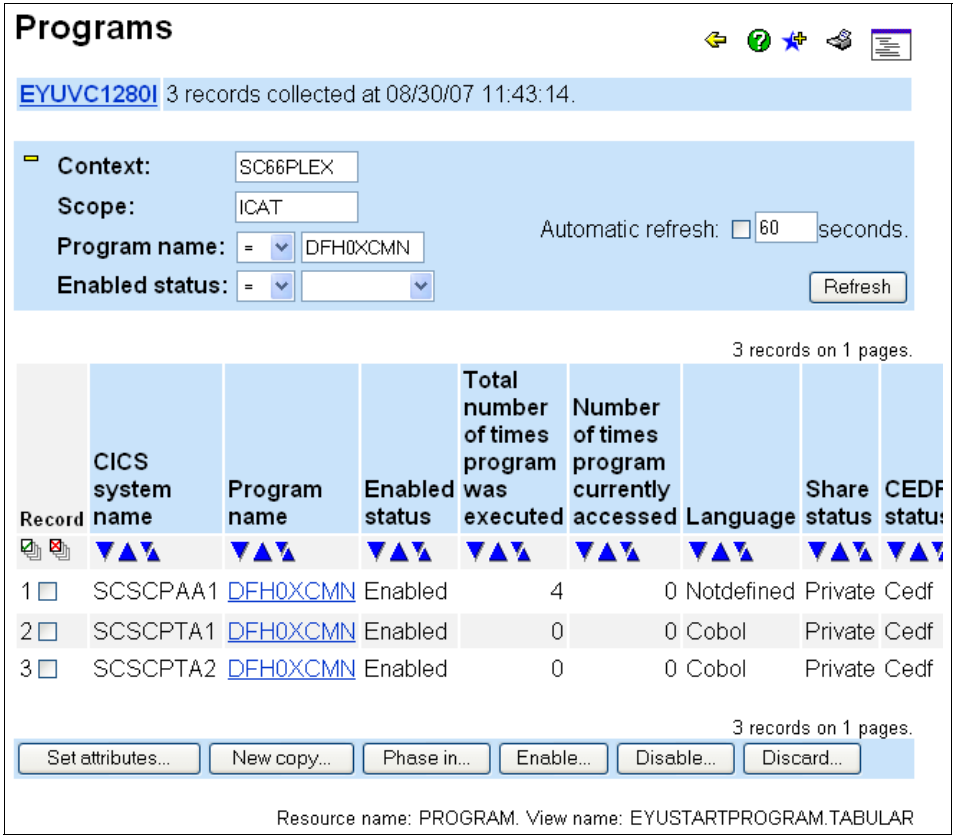


Figure 5-50 Detailed program view

Select the programs to be refreshed and click **Phase in** (Figure 5-51).

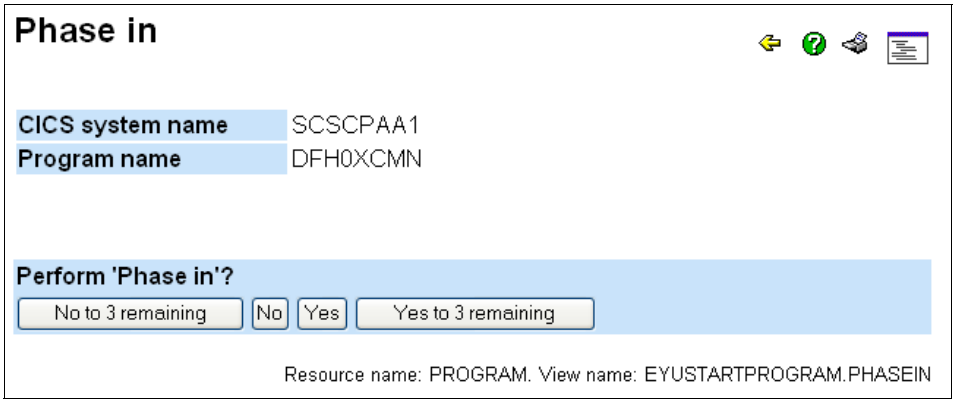


Figure 5-51 Phase in confirmation

Click **Yes to 3 remaining** to refresh the program in all the regions (Figure 5-52).

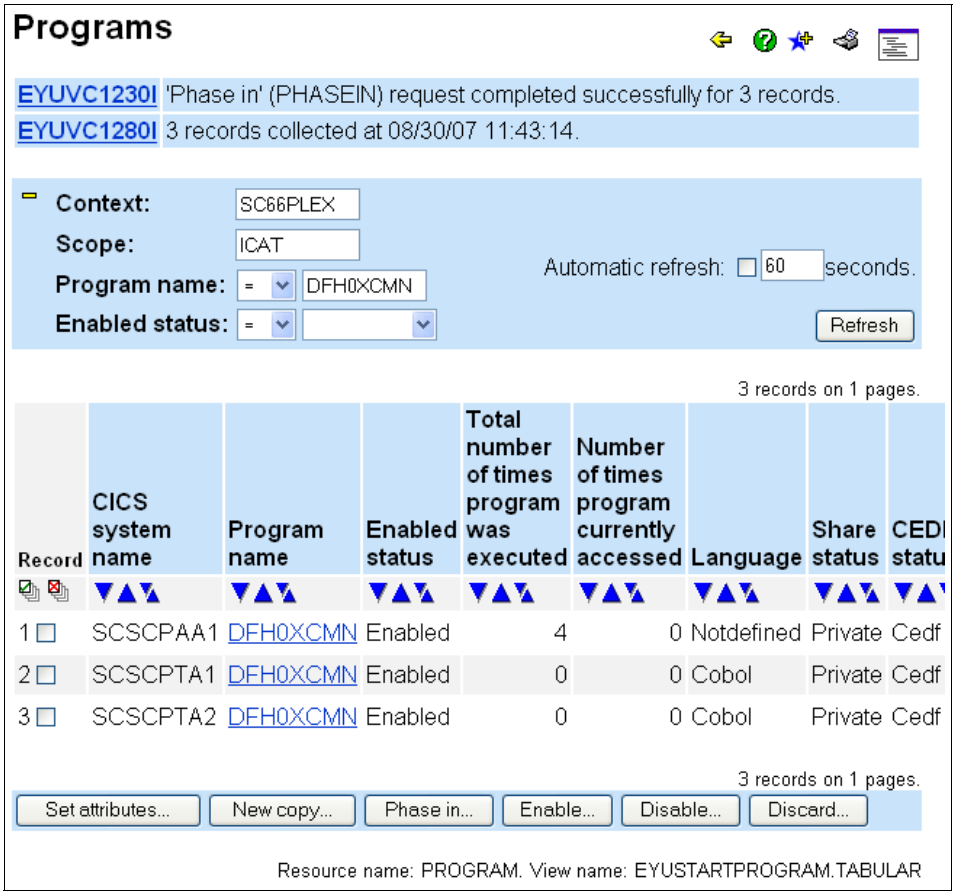


Figure 5-52 Programs have been refreshed

6

Configuring CICS as a service requester

In this chapter, we show how we set up a CICS Web services requester application for high availability. For this purpose, we use the CICS sample catalog application. We look at the following topics:

- The configuration
- The CICS definitions
- Changes to the application

6.1 Configuration update

We want to separate CICS Web services from the business logic. To achieve this goal, we set up two CICS systems to run the business logic programs and two CICS Systems dedicated to run all CICS Web services requests. The CICS Web service requests are invoked by the business logic programs using a distributed program link to a wrapper program running in the CICS Web service requester regions. To workload balance the DPL requests to the wrapper program, we use CICSPlex SM.

This configuration has a number of advantages. These are:

- You do not have to define WEBSERVICEs and PIPELINES in all the AORs.
- It is always a best practice to clearly distinguish the roles of different CICS regions.
- The Distributed Program Links from application regions can be workload managed by CICSPlex SM across a set of Web service requester regions, thus providing workload balancing between LPARs for higher availability.
- You may not want the AORs to have access to external resources through TPC/IP.
- You can run the business application in a region that is not running CICS TS V3.1 or higher.

In order to test this configuration, we modify our configuration by providing a WebSphere Application Server on a Windows® machine to act as the Web service provider. We then deploy the sample Web service provider application shipped with CICS as ExampleAppDispatchOrder.ear to the WebSphere Application Server.

Figure 6-1 on page 163 shows the configuration we use to test the outbound option.

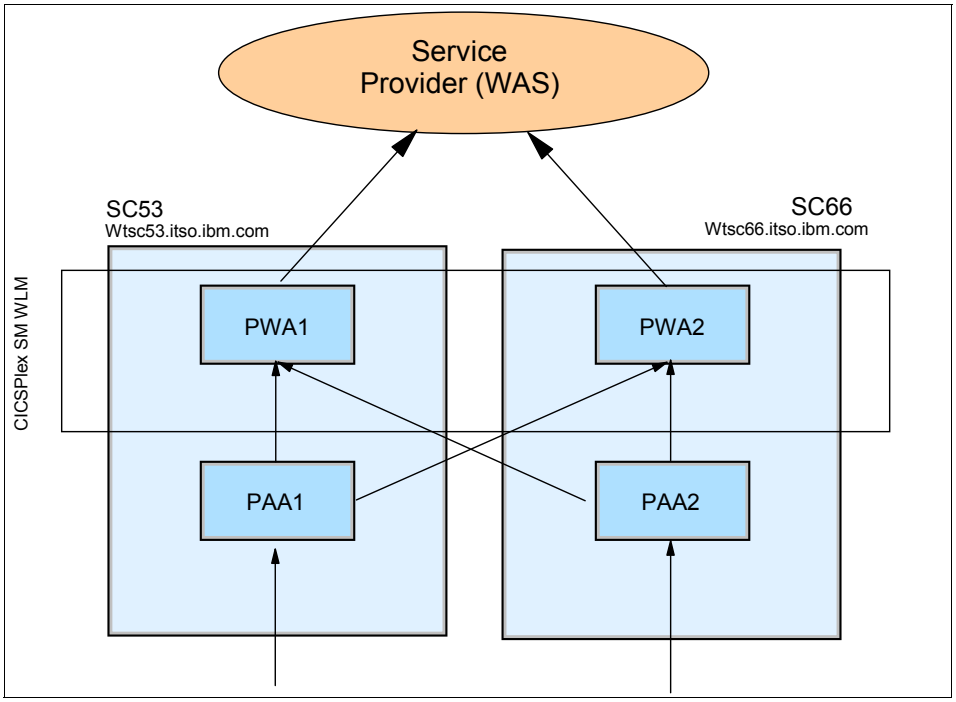


Figure 6-1 CICS Web services requester configuration

6.1.1 Resource definitions

We first install a PIPELINE definition for the Web service requester in both regions PWA1 and PWA2. We use the CICS-provided basicsoap11requester.xml as our pipeline configuration file after copying it into a private directory.

Figure 6-2 shows this PIPELINE definition.

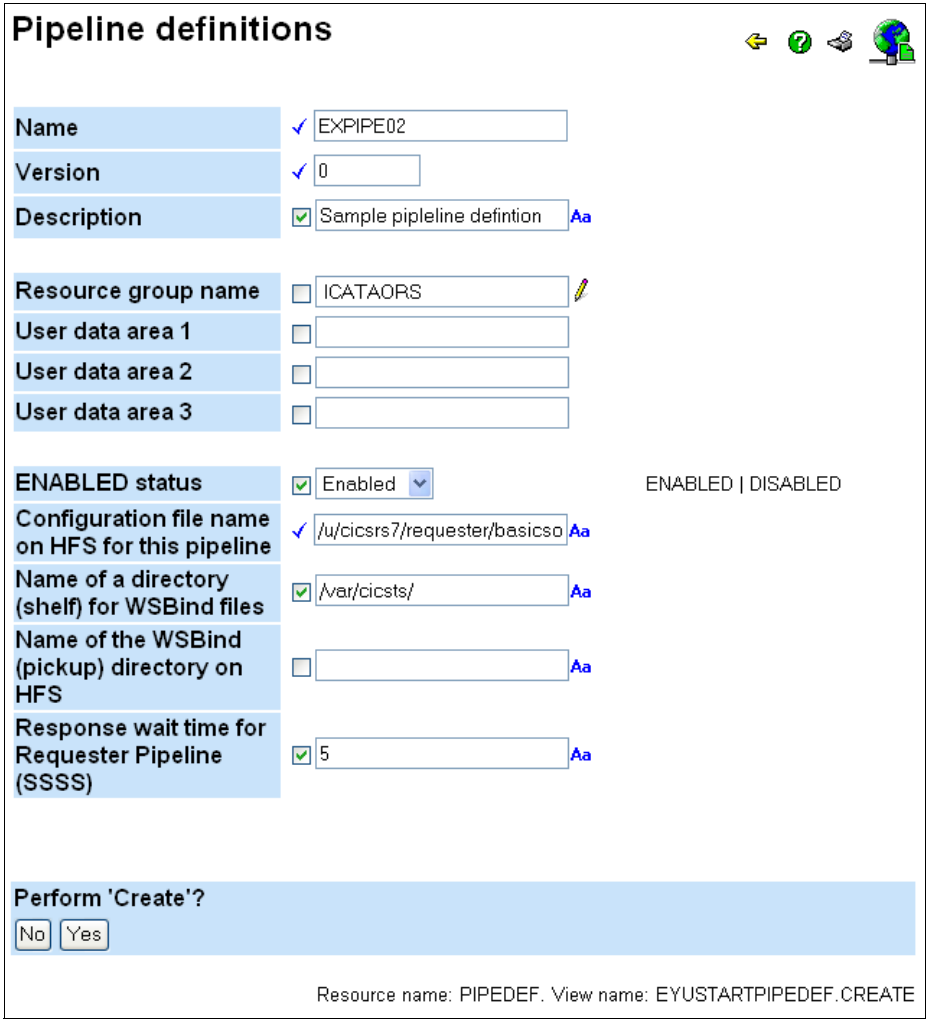


Figure 6-2 Pipeline definition for the service requester Web service

The configuration file name is the name of the HFS file, where we place the CICS-provided basicsoap11requester.xml pipeline configuration file.

We define five seconds as the maximum time we will wait for the Web service provider to respond. Of course, this value should be evaluated based on the application and the Web service provider.

Note: If you use the CICS-provided pipeline configuration file basicsoap11provider.xml, do not forget to change the encoding parameter specified on the first line of the file if your Web service provider runs within WebSphere. It has to be changed from encoding="EBCDIC-CP-US"?> to encoding="UTF-8"?>.

We decide to use a static definition for our WEBSERVICE instead of doing pipeline scanning, because we think this is more appropriate for a production environment. Consequently, we change the name of the WEBSERVICE that is provided as a parameter to the EXEC CICS INVOKE WEBSERVICE command in the dispatch manager program, DFH0XWOD, to the one we chose for our Web service (DISPORDR). The reason for this is that the pipeline scanning process generates a WEBSERVICE resource definition with the name of the actual Web service. In our case, the generated name would be dispatchOrder. This name is more than eight characters. CICS will not allow this, so we cannot use it for hardcoded definitions. Because DFH0XWOD is designed for pipeline scanning, it uses dispatchOrder, but in our case we use DISPORDR.

See “Application changes” on page 168 for details on changing the application.

Note: If you use pipeline scanning in your development/test environment and use hardcoded definitions in your production environment, you might have to make a similar change before you move your application to production. Alternatively, you may limit the length of Web service names to eight characters.

We then define the WEBSERVICE shown in Figure 6-3.

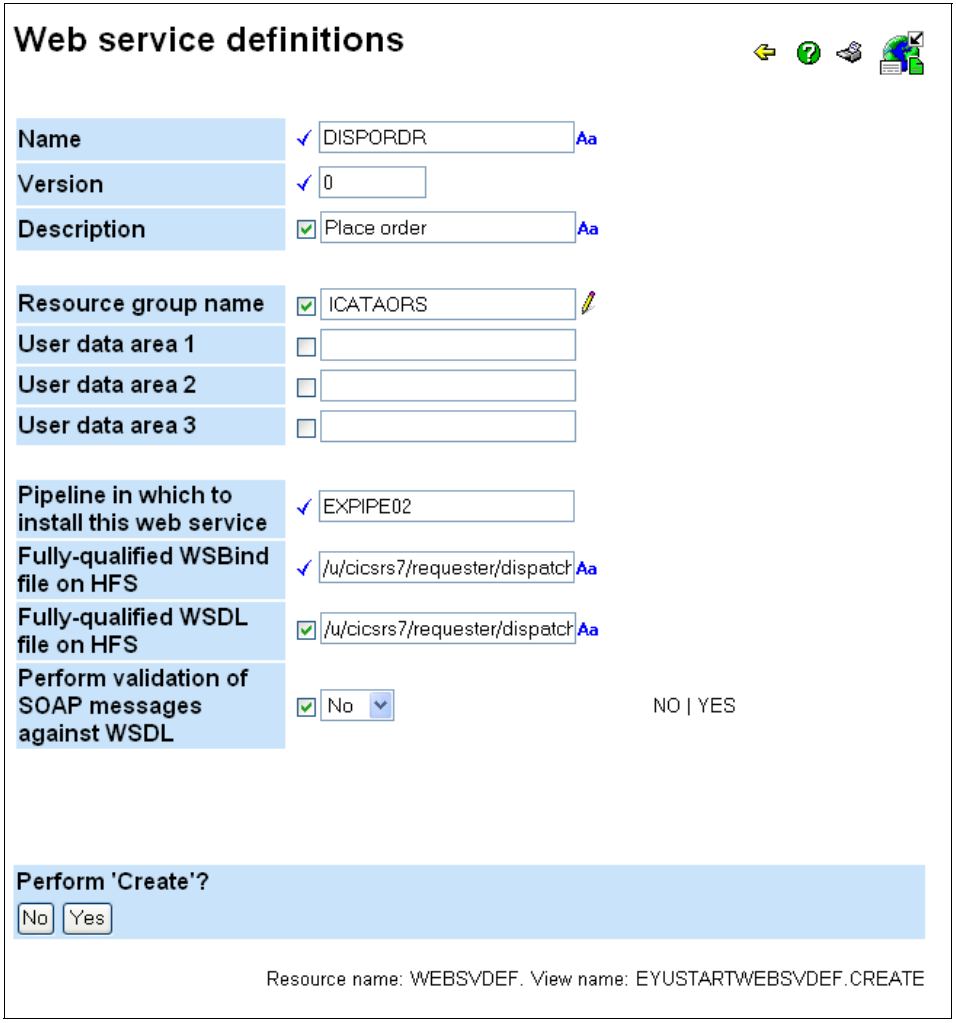


Figure 6-3 WEBSERVICE resource definition

In order to dynamically route the EXEC CICS LINK to ABCXWO2, we define this program as remote in PAA1 and PAA4. In PWA1 and PWA2, this program is autoinstalled.

We also define transaction XXXX in PWA1 and PWA2 with the initial program as DFHMIRS. This is because it will be used as a mirror transaction for DPL.

Updating the URL in the configuration file

In order to use the CICS example application, you have to set up the EXMPCONF configuration file. You have to specify that you want to use outbound Web services. You also have to specify the Web service provider URI. In order to do this task, we should refer to Chapter 3, “Web services using HTTP”, in *Implementing CICS Web Services*, SG24-7206. Alternatively, you can import the ExampleAppDispatchOrder.ear file into WebSphere Application Server Toolkit, Rational® Application Developer, or WebSphere Developer for System z and look at the WSDL file that was published by this service.

Select **ExampleAppDispatchOrderWeb** → **Web Content** → **wsdl** and click all the plus signs underneath. Open the dispatchOrder.wsdl file and look at the soap:address entry in the service element to discover that the correct URL to use is `http://WAS server address:port/exampleApp/services/dispatchOrderPort`, as shown in Figure 6-4.

Figure 6-4 dispatchOrder.wsdl SOAP address location

This URL is a parameter to an EXEC CICS INVOKE WEBSERVICE command that the dispatch manager program (DFH0XWOD) uses.

Application changes

In order to test this configuration, we split the dispatch manager program of the catalog manager application, DFH0XWOD, into two programs:

- ABC0XWO1 runs in the business logic regions and LINKs to ABC0XWO2, passing a container with dispatch order request data. Another container that contains working variables for the INVOKE WEBSERVICE command is passed on the same channel.
- ABC0XWO2 runs in the CICS Web service requester regions and gets the working variables from the container that it gets on the channel and issues an INVOKE WEBSERVICE command. The output data and return code are passed back to ABC0XWO1 in another container.

In order to invoke the new dispatcher, we use the ECFG transaction to change the EXMPCONF file. We change the name of the Order Dispatch WebService from DFH0XWOD to ABC0XWO1. We also change the outbound service to YES and the URI to match our configuration.

Note: In order to enter the URI in mixed case, we switch off upper case translation for the terminal using CEOT NOUCTRAN.

Example 6-1 shows the updated configuration screen of the ECFG transaction.

Example 6-1 Order Dispatch WebService program changed

```
CONFIGURE CICS EXAMPLE CATALOG APPLICATION

           Datastore Type ==> VSAM               STUB|VSAM
     Outbound WebService? ==> YES                 YES|NO
          Catalog Manager ==> DFHOXCMN
          Data Store Stub ==> DFHOXSDS
          Data Store VSAM ==> DFHOXVDS
      Order Dispatch Stub ==> DFHOXSOD
Order Dispatch WebService ==> ABCOXWO1
            Stock Manager ==> DFHOXSSM
           VSAM File Name ==> EXMPCAT
  Server Address and Port ==>
  Outbound WebService URI ==> http://9.42.170.163:9080/exampleApp/services
                          ==> /dispatchOrderPort
                          ==>
                          ==>
                          ==>
                          ==>
APPLICATION CONFIGURATION UPDATED
PF             3 END                                                12 CNCL
```

Dynamic routing

We set up CICSPlex SM to dynamically route the DPL requests from the business logic program (ABC0XWO1) running in PAA1 and PAA2 to the wrapper program (ABC0XWO2) running in PWA1 and PWA2. See Chapter 5, “Configuring CICS as a service provider” on page 103 for more details.

We want to be sure that the transaction ID of the mirror tasks running in the Web service requester regions were the same as the transaction ID of the task that did the link. In order to do that task, we use a XCPREQ global user exit. This exit is invoked when the business logic program issues the EXEC CICS LINK and sets the TRANSID in the link to the same as the transaction ID of the current task. Of course, this could also have been achieved by specifying TRANSID(EIBTRNID) in the EXEC CICS LINK statement. See Appendix D, “Sample XPCREQ global user exit” on page 251 for more details.

6.2 Using multiple Web service providers

Until now, we have only been using a single Web service provider. To avoid this provider from becoming a single point of failure, we want to add another Web service provider.This would also give us the possibility to make some sort of workload balancing across the Web service providers.

These instances can be on different distributed boxes (AIX®, Linux®, Windows, for example), or on different z/OS LPARs. If your Web service providers run in a Parallel Sysplex as your CICS regions, you might consider using Sysplex Distributor for balancing workload across these regions.

We clone the Web service provider by setting up another Windows machine with the same configuration as the first one. Figure 6-5 shows this new configuration.

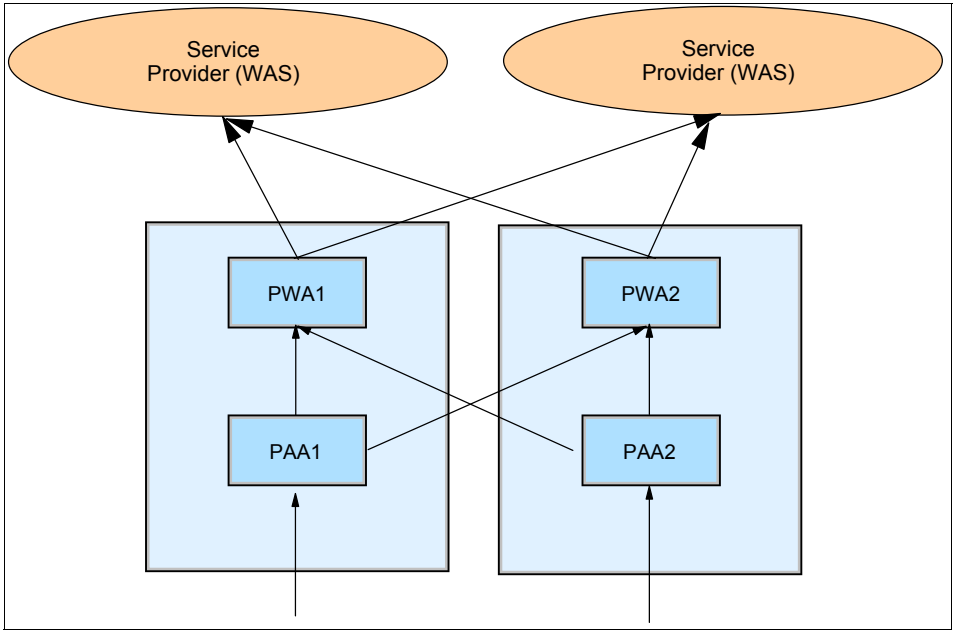


Figure 6-5 Web service configuration

Resource definitions

Since the WEBSERVICEs and PIPELINEs are the same as in the single provider configuration, there are no changes to resource definitions.

Application changes

If the Web service providers are not running in the same Sysplex as the Web service requester regions, you have to implement a programmed workload balancing and failure handling feature.

A CICS program that issues Web service requests (regardless of whether it runs in Web service region or in an application region) can be designed to issue EXEC CICS INVOKE WEBSERVICE requests to different Web service providers following a certain algorithm, let us say, round-robin. To be able to do this task, the program must specify the URI parameter on the EXEC CICS INVOKE WEBSERVICE.

In addition, the program may be coded in such a way that if there is a failure of the EXEC CICS INVOKE WEBSERVICE request against one provider, the same request is issued against another provider. One possible condition that should be dealt with is the timeout, DFHRESP(TIMEDOUT). See 2.4.2, “Processing the outbound service request” on page 30 for a description of how to specify timeout for outbound requests (the RESPWAIT parameter on a pipeline definition).

If you use these techniques, you achieve:

- Workload balancing between several Web service providers that provide the same service.
- Programmed failover capability in case one of the Web service providers has a problem; it may be too busy to service your request from CICS in time.

In order to test these ideas, we make additional changes to the catalog manager application:

- We change the artifacts of the ECFG transaction, the DFH0XS3 mapset, the DFH0XCFG program, and the EXMPCONF file to allow specification of the URI of an additional Web service provider. The updated configuration screen is shown in Example 6-2 on page 172.
- We change the record length of the EXMPCONF file.
- We change the LENGTH attribute of the READ command in the VSAM data handler program DFH0XVDS and in our dispatch manager program ABC0XWO1.
- We change lengths of various data areas in DFH0XVDS, ABC0XWO1, and ABC0XWO2 to allow specification of the URI of the second Web service provider.
- We change the logic of ABC0XWO2 program in such a way that it now uses a round-robin algorithm to send EXEC CICS INVOKE WEBSERVICE requests alternately to our two Web service providers. The A-B switch that controls who is the next provider is kept as a temporary storage queue item. Example 6-3 on page 172 shows the code that implements the round-robin logic.
- We introduce a retry routine. In case there is a failure of an INVOKE WEBSERVICE command, we try to use the other provider. Only in the case where service requests to both providers fail would we return an error message to the business logic program. Example 6-4 on page 174 shows the retry routine of ABC0XWO2.
- Finally, we make sure that outbound requests go to both providers in turn by using CEDF and the SHOWINFO message handler program.

Example 6-2 Defining an additional Web service provider

```
CONFIGURE CICS EXAMPLE CATALOG APPLICATION

           Datastore Type ==> VSAM               STUB|VSAM
     Outbound WebService? ==> NO                 YES|NO
          Catalog Manager ==> DFHOXCMN
          Data Store Stub ==> DFHOXSDS
          Data Store VSAM ==> DFHOXVDS
      Order Dispatch Stub ==> DFHOXSOD
Order Dispatch WebService ==> ABCOXWO1
            Stock Manager ==> DFHOXSSM
           VSAM File Name ==> EXMPCAT
  Server Address and Port ==>
  Outbound WebService URI ==> http://9.42.170.163:9080/exampleApp/services
                          ==> /dispatchOrderPort
                          ==>
                          ==>
                          ==>
                          ==>
  Outbound WebService URI ==> http://9.42.170.164:9080/exampleApp/services
                          ==> /dispatchOrderPort


APPLICATION CONFIGURATION UPDATED
PF             3 END                                                  12 CNCL
```

Example 6-3 The round-robin logic of ABC0XWO2

```
*-------------------------------------------------------------*
* Here is the logic to choose the outbound request destination
*-------------------------------------------------------------*

          MOVE 1 TO WS-ITEM-LENGTH
          MOVE 1 TO WS-ITEM-NUMBER
     EXEC CICS READQ TS QUEUE(WS-TSQNAME) INTO(WS-ABSWITCH)
              LENGTH(WS-ITEM-LENGTH) ITEM(WS-ITEM-NUMBER)
              RESP(RESP)
     END-EXEC.
     EVALUATE RESP
        WHEN DFHRESP(QIDERR)
          MOVE 'A' TO WS-ABSWITCH
          MOVE 1 TO WS-ITEM-LENGTH
          MOVE 1 TO WS-ITEM-NUMBER
          EXEC CICS WRITEQ TS QUEUE(WS-TSQNAME) FROM(WS-ABSWITCH)
               LENGTH(WS-ITEM-LENGTH) ITEM(WS-ITEM-NUMBER)
          END-EXEC
     END-EVALUATE.
```

```
*-------------------------------------------------------------*
* Make the Invoke call
*-------------------------------------------------------------*
EVALUATE WS-ABSWITCH
          WHEN 'A'
             MOVE 'B' TO WS-ABSWITCH
             EXEC CICS INVOKE WEBSERVICE(WS-WEBSERVICE-NAME)
                   CHANNEL(WS-CHANNELNAME)
                   URI(WS-ENDPOINT-URI)
                   OPERATION(WS-OPERATION)
                   RESP(RESP) RESP2(RESP2)
             END-EXEC
          WHEN 'B'
             MOVE 'A' TO WS-ABSWITCH
             EXEC CICS INVOKE WEBSERVICE(WS-WEBSERVICE-NAME)
                   CHANNEL(WS-CHANNELNAME)
                   URI(WS-ENDPOINT2-URI)
                   OPERATION(WS-OPERATION)
                   RESP(RESP) RESP2(RESP2)
             END-EXEC
      END-EVALUATE




* Check the return code was normal

      EVALUATE RESP
          WHEN DFHRESP(NORMAL)
              MOVE 1 TO WS-ITEM-LENGTH
MOVE 1 TO WS-ITEM-NUMBER
             EXEC CICS WRITEQ TS QUEUE(WS-TSQNAME)
             FROM(WS-ABSWITCH)
                  LENGTH(WS-ITEM-LENGTH) ITEM(WS-ITEM-NUMBER)
                  REWRITE
             END-EXEC

             EXEC CICS GET CONTAINER(WS-SERVICE-CONT-NAME)
                       CHANNEL(WS-CHANNELNAME)
                       INTO(CA-ORD-RESPONSE-MESSAGE)
             END-EXEC

         WHEN OTHER
             PERFORM RETRY-INVOKE
     END-EVALUATE.
```

Example 6-4 The RETRY-INVOKE routine of ABC0XWO2

```
*-------------------------------------------------------------*
 RETRY-INVOKE.
     EVALUATE WS-ABSWITCH
         WHEN 'A'
            EXEC CICS INVOKE WEBSERVICE(WS-WEBSERVICE-NAME)
                  CHANNEL(WS-CHANNELNAME)
                  URI(WS-ENDPOINT-URI)
                  OPERATION(WS-OPERATION)
                  RESP(RESP) RESP2(RESP2)
            END-EXEC
         WHEN 'B'
            EXEC CICS INVOKE WEBSERVICE(WS-WEBSERVICE-NAME)
                  CHANNEL(WS-CHANNELNAME)
                  URI(WS-ENDPOINT2-URI)
                  OPERATION(WS-OPERATION)
                  RESP(RESP) RESP2(RESP2)
            END-EXEC
     END-EVALUATE

* Check the return code was normal

     EVALUATE RESP
         WHEN DFHRESP(NORMAL)
             MOVE 1 TO WS-ITEM-LENGTH
MOVE 1 TO WS-ITEM-NUMBER
             EXEC CICS WRITEQ TS QUEUE(WS-TSQNAME)
             FROM(WS-ABSWITCH)
                  LENGTH(WS-ITEM-LENGTH) ITEM(WS-ITEM-NUMBER)
                  REWRITE
             END-EXEC
             EXEC CICS GET CONTAINER(WS-SERVICE-CONT-NAME)
                       CHANNEL(WS-CHANNELNAME)
                       INTO(CA-ORD-RESPONSE-MESSAGE)
             END-EXEC
             MOVE ZERO TO CA-ORD-RETURN-CODE

         WHEN DFHRESP(INVREQ)
             MOVE 'Error calling dispatch service - INVREQ'
                 TO CA-ORD-RESPONSE-MESSAGE
             MOVE 30 TO CA-ORD-RETURN-CODE

         WHEN DFHRESP(NOTFND)
             MOVE 'Error calling dispatch service - NOT FOUND'
                 TO CA-ORD-RESPONSE-MESSAGE
             MOVE 31 TO CA-ORD-RETURN-CODE
```

```
         WHEN DFHRESP(TIMEDOUT)
MOVE 'Error calling dispatch service - TIMED OUT'
                 TO CA-ORD-RESPONSE-MESSAGE
             MOVE 32 TO CA-ORD-RETURN-CODE

         WHEN OTHER
             MOVE 'Error calling dispatch service'
                 TO CA-ORD-RESPONSE-MESSAGE
             MOVE 32 TO CA-ORD-RETURN-CODE
     END-EVALUATE.

     EXIT.
```

6.3 Summary

Our outbound configuration utilizes the workload management capabilities of CICSPlex SM. We use multiple CICS application and Web service requester regions, therefore we are not dependent on a single CICS application region or a single Web serviced requester region.

We use multiple cloned Web service providers so we can implement workload balancing logic in our application. We are not dependent on the availability of a specific Web service provider.

Furthermore, we are not dependent on the availability of an z/OS image, because we are running on two z/OS images.

Our environment is scalable. We are able to add more CICS application regions, more Web serviced requester regions, and more WebSphere Application Server Web service providers.

7

CICS Web services in the banking industry

In this chapter, we describe a practical example of a CICS Web services implementation based on a proof of concept with a large financial group carried out in the IBM Customer Support Center at Montpellier, France.

We describe the solution that we designed and explain how it meets the specific requirements of the customer, including the workload management and availability requirements. We discuss some of the design decisions that were made based on the customers requirements and provide a description of the CICS infrastructure that was created to test the solution.

The tested infrastructure was based on CICS Transaction Server, CICSPlex SM, WebSphere Application Server, Parallel Sysplex, the WebSphere DataPower® SOA Appliance, and Tivoli monitoring.

7.1 Project introduction

We begin by providing an overview of the project and by providing background information about the specific customer goals. We then outline the target architecture and highlight the main functional and non-functional requirements.

7.1.1 Customer goals

The customer is a large European financial services group that incorporates several different well known brands. The core banking applications run by the customer are based on CICS.

Note: A brand is a subsidiary of the financial group (for example, a retail bank or insurance company).

Figure 7-1 shows the customer organization that handles business service requests from a number of different requesters, including branch offices, customers, suppliers, and trusted partners.

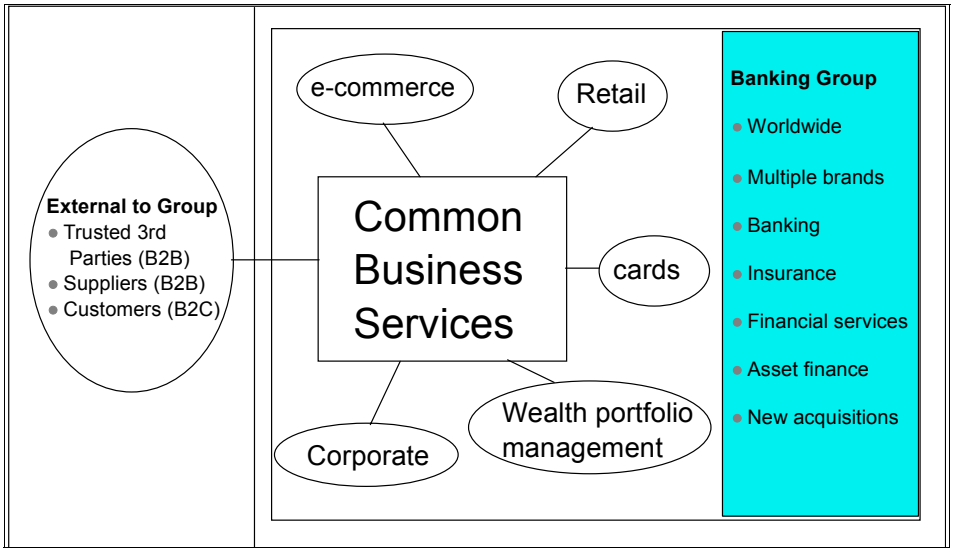


Figure 7-1 Common business services

Most of the common business services exist already as CICS functions that are part of the core banking applications. These functions are called today using a number of different access methods including terminal emulators and connectors, such as the CICS Transaction Gateway.

7.1.2 Target service-oriented architecture

The proposed CICS Web Services infrastructure will provide business services to the group as a whole; sometimes individual brands will use the same business service, sometimes those individual brands will have specific unique requirements. The target architecture (Figure 7-2) allows CICS functions to be published as services and also allows CICS programs to make service calls to non-CICS services.

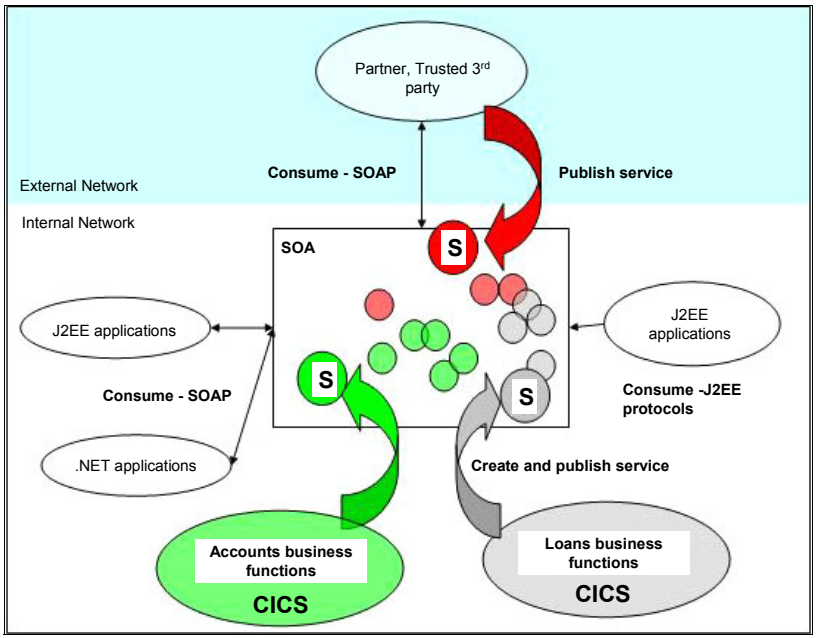


Figure 7-2 Target service-oriented architecture

Figure 7-2 shows how it is intended for CICS services, for example, Accounts and Loans services, to be reused across the group:

- To publish CICS services to internal group systems for consumption using J2EE protocols
- To publish CICS services to internal group systems for consumption using SOAP-based protocols
- To publish CICS services to external partners for consumption using SOAP-based protocols
- To allow external partners to publish their own services for consumption by internal CICS systems using SOAP-based protocols

An example of an Accounts service is the retrieval of customer account information in which the service requester provides an account number and is returned account information, such as customer name, address, balance, and overdraft limit.

7.1.3 Solution requirements

It is the main functional requirements of the project that led to the choice of using CICS Web Services support for this project. Table 7-1 shows how CICS Web services support meets the major functional requirements of the project.

Table 7-1 Main functional requirements

Functional requirement	Description
Ability to invoke CICS Web services from any platform (including J2EE and .Net)	Web services are platform neutral.
Ability to distinguish requests for different brands based on URI	The URIMAP resource definition can be used to filter requests based on URI.
Bi-directional support (CICS can be a service provider or service requester.)	CICS Web services support is bi-directional.
Support for long messages	CICS Web services support is not subject to the 32 KB message length limit imposed by other connection options.
Ability to publish CICS services to a service registry	CICS Web services can be published to the WebSphere Services Registry and Repository (WSRR).

The main non-functional requirements of the project are:

- High availability

  The business-critical Web services must be available at all times. The infrastructure must be able to support continuous service availability across planned and unplanned outages.

- Security

  The solution must be secure and must meet the end-to-end security requirements of the customer. A basic security requirement is that the service requester's identity must flow with the message and that for certain services (for example, bank account transfers) the target CICS transaction must run with the requester's identity.

  See 7.1.6, “Security model” on page 185 for a description of the security model used in the project.

- Performance and scalability

  The solution must meet the performance and scalability expectations of the customer. One of the prime objectives is to demonstrate how a Web services workload can be dynamically managed based on performance goals and to show that a CICS Web services infrastructure can handle the kind of workload currently supported with other access methods.

- Real-time monitoring

  Problems must be automatically highlighted, and detailed real-time information (including service hit rates, response times, and message lengths) must be available for problem diagnosis.

The subsequent sections of this chapter focus on how the non-functional requirements were addressed in our project.

7.1.4 Usage scenarios

CICS Web services may be accessed directly from a service requester or indirectly through a service bus. A service bus approach has several advantages:

- Service requesters do not need to have knowledge of the specific location of service providers.
- The service bus can manage the security characteristics of inbound or outbound Web services requests, changing transports, and performing security tasks, such as authentication and identity mapping, and asserting identities to the target service provider
- The service bus can simplify the management of service deployment and configuration.

While there are a number of service bus implementations that meet the above requirements, the choice was made in this project to use a WebSphere SOA DataPower Appliance as a service bus component.

Note: IBM WebSphere DataPower SOA appliances are purpose-built, easy-to-deploy network devices that simplify and accelerate XML and Web services deployments.

There are three types of DataPower appliance available, each building on the features of the last:

- IBM WebSphere DataPower XML Accelerator XA35

  Accelerates common types of XML processing by offloading this processing from servers and networks. It can perform XML parsing, XML Schema validation, XPath routing, Extensible Stylesheet Language Transformations (XSLT), XML compression, and other essential XML processing with wirespeed XML performance.

- IBM WebSphere DataPower XML Security Gateway XS40

  Provides a security-enforcement point for XML and Web services transactions, including encryption, firewall filtering, digital signatures, schema validation, WS-Security, XML access control, XPath, and detailed logging.

- IBM WebSphere DataPower Integration Appliance XI50

  Transport-independent transformations between binary, flat text files, and XML message formats. Visual tools are used to describe data formats, create mappings between different formats, and define message choreography.

For full product information about IBM WebSphere DataPower SOA Appliances, refer to:

`http://www-306.ibm.com/software/integration/datapower/index.html`

A DataPower appliance can be used in conjunction with CICS Web services to help secure the services and to offload expensive operations by processing the complex part of XML messages (such as an XML signature) at wirespeed.

WebSphere DataPower Integration Appliance XI50 is used in this project for the following reasons:

- There is no immediate requirement to support a large diversity of message formats and protocols (messages are XML based and HTTPS is the transport protocol).
- The mediation requirements are relatively simple (identity mapping, auditing, routing, and XML transformation).
- DataPower provides a high performance, cost efficient execution of translation and routing instructions.

Note that the use of DataPower as a service bus component does not rule out the use of a more centralized ESB (Enterprise Service Bus) solution for the future, for example, using the WebSphere ESB. These different ESB solutions can be combined into a “federated” ESB.

The following usage scenarios were tested for this project:

- CICS as a service provider

  Services may be accessed directly, for example, from internal service requesters running in WebSphere Application Server, or indirectly through DataPower from internal .NET service requesters or external trusted third parties (see Figure 7-3).

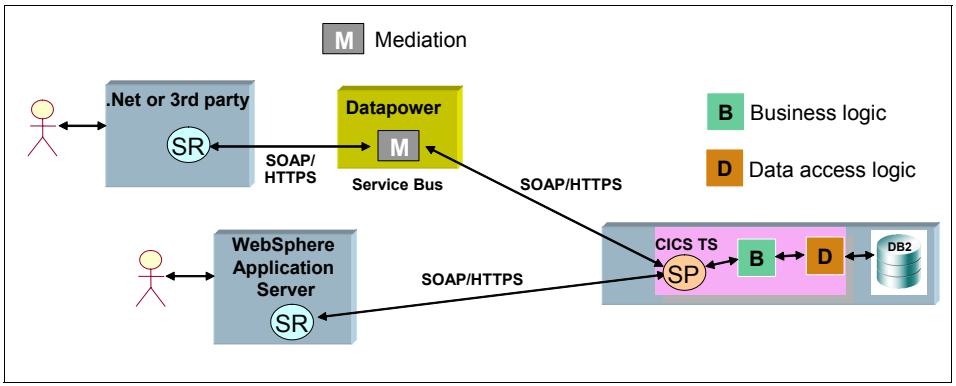


Figure 7-3 CICS service provider scenarios

Figure 7-3 shows a CICS service provider (SP) application being invoked by different service requester (SR) applications. The target CICS program is a business logic program that then calls a data access program to access a DB2 database. A Mediation (M) is used in DataPower to perform identity mapping, auditing, routing, and (in selective cases) XML transformation.

Note: In this scenario, DataPower receives and sends SOAP meesages. Other usage scenaios are also possible, for example, DataPower is capable of converting a SOAP message into a CICS COMMAREA and sending the COMMAREA to CICS using MQ.

- CICS as a service requester

  When CICS is a service requester, CICS applications need access to services in WebSphere Application Server and also other services that will be accessed indirectly through DataPower (see Figure 7-4).

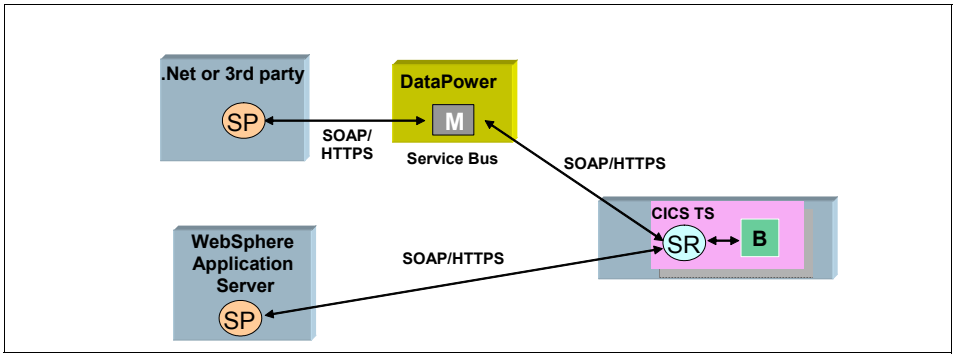


Figure 7-4 CICS service requester scenarios

Figure 7-4 shows a CICS service requester application sending requests to different service provider applications.

7.1.5 Bottom-up, top-down, or meet in the middle

Before service enabling a CICS application, you have to decide which style of Web service to use (bottom-up, top-down, or "meet in the middle").

The starting point for the bottom-up approach is the CICS business logic program. The CICS Web services assistant utility DFHLS2WS is used to create WSDL and wsbind files based on the language structure of the business logic program.

The starting point for the top-down approach is the WSDL file. The CICS Web services assistant utility DFHWS2LS is used to create the wsbind file and a language structure for a new CICS business logic program.

We chose the meet in the middle approach because it offers the best flexibility and control over XML message formats. Using this approach, DFHWS2LS is used to create the wsbind file and a language structure for a wrapper program. The wrapper program acts as the service provider application and, based on the SOAP request message, it creates a COMMAREA (or Container) for the target business logic program.

An advantage of the meet in the middle approach is that it can be used when the COMMAREA interface of an existing business logic program contains unsupported fields or has a complexity that would create an ineffecient data mapping. It also facilitates the creation of a high availability configuration.

A wrapper program can be used for both service provider and service requester scenarios:

- For a CICS service provider, the wrapper program is used as the Web service implementation. The wrapper program then links to the existing business logic program.
- For a CICS service requester, an existing business logic program links to a wrapper program. The wrapper program uses the EXEC CICS INVOKE WEBSERVICE API to invoke a Web service.

The availability advantage of the wrapper approach is that it allows the wrapper program to be deployed in a different CICS region than the existing business logic programs. This makes it possible to service enable a CICS program running in a back level CICS region and it is also consistent with the principal of configuring CICS regions with different roles within a CICSplex. See 7.3.1, “CICSplex configuration” on page 198 for information about how we deployed CICS Web services into a CICSplex.

7.1.6 Security model

The end-to-end security requirement is satisfied using a security solution based on *identity assertion*. Identity assertion is an authentication mechanism that is applied among three parties: a client, an intermediary server, and a target server.

When CICS is acting as a service provider, the security flow is as follows:

- The intermediary server (for example, WebSphere Application Server or a DataPower SOA appliance) authenticates the client, and transfers the client's identity as a SOAP Security header with the request message sent to the target CICS server. Example 7-1 is an example of a Security header.

Example 7-1 Security header

```
<soapenv:Header>
  <wsse:Security soapenv:mustUnderstand="1"
xmlns:wsse="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wss
ecurity-secext-1.0.xsd">
    <wsse:UsernameToken>
      <wsse:Username>NIGEL2</wsse:Username>
    </wsse:UsernameToken>
  </wsse:Security>
```

```
</soapenv:Header>
```

Before propagating the client's identity, the intermediate server maps the identity to a RACF identity recognized by CICS. This can be done using LDAP or by making a call to a security manager, such as IBM Tivoli Federated Identity Manager.

- The intermediate server must establish a trust relationship with CICS by authenticating itself and then by being recognized as a trusted partner of the CICS region. In this implementation, the trust relationship between the intermediate server and CICS is established using SSL client authentication.
- For high value services (for example, Account transfers), a user written CICS header processing program is invoked as part of the service provider pipeline processing:
  - It parses the Security header.
  - It extracts the RACF user ID from the Security header and writes it to the DFHWS-USERID container. This then causes a context switch to occur when invoking the wrapper program so that the wrapper program runs with the asserted identity.

    CICS does the normal authorization checks, thus making sure that the user is authorized to use this particular service.
  - Writes an audit record to a user journal.

  For other requests (for example, Account inquiries), the wrapper program runs with a function ID. The function ID is the identity of the server that sends the request to CICS.

A similar end-to-end security solution based on identity assertion is used for the CICS service requester scenario. In this case, a user written CICS header processing program, which is invoked as part of the service requester pipeline processing, obtains the RACF user ID associated with the CICS task and sends this as a Security token to the intermediate server.

Identity assertion is an extended WS-Security mechanism supported by WebSphere Application Server Version V6 and WebSphere DataPower.

Note: The WS-Security support provided with CICS TS V3.2 enables identity assertion without the need to write a header processing program. However, this project used CICS TS V3.1 and therefore we needed to write a header processing program to enable identity assertion.

7.2 CICS definitions

In this section, we look at the main CICS resource definitions that are made to support the service provider and service requester usage scenarios. Note that we only provide information about the resource definitions related to the Web services and not all the CICS resources used in the scenarios.

7.2.1 CICS as a service provider

We look at the CICS definitions for the Account transfer and Account inquiry services for brand ABC.

Account transfer service

Here are the set of CICS resource definitions used for processing a Web service request with the URL `https://abc.pssc.fr:20002/Accounts/AccountTransfer`. The structure of th URL is as follows:

https	The protocol used to invoke the service.
abc.pssc.fr	The host name used to invoke all brand ABC services.
20002	The TCP/IP port used for brand ABC services.
Accounts	Specfies that the request is for the Accounts application.
AccountTransfer	Specifies that the request is for the Account transfer service.

TCPIPSERVICE

The TCPIPSERVICE is used to specify the transport requirements for the service, in particular, that the service is only accessible through an SSL client authenticated connection. The TCPIPSERVICE is shown in Figure 7-5.

```
OVERTYPE TO MODIFY                                        CICS RELEASE = 0640
CEDA DEFine TCpipservice( TCPIPABC )
    TCpipservice   : TCPIPABC
    GROup          : WSIGOR
    DEscription  ==> TCPIPSERVICE FOR BRAND ABC
    Urm          ==> DFHWBADX
    POrtnumber   ==> 20002              1-65535
    STatus       ==> Open               Open ! Closed
    PROtocol     ==> Http               Iiop ! Http ! Eci ! User
    TRansaction  ==> CWXN
    Backlog      ==> 00005              0-32767
    TSqprefix    ==>
    Ipaddress    ==>
    SOcketclose  ==> 000030             No ! 0-240000 (HHMMSS)
    Maxdatalen   ==> 000032             3-524288
  SECURITY
    SSl          ==> Clientauth         Yes ! No ! Clientauth
```

Figure 7-5 TCPIPSERVICE TCPIPABC

- PORTNUMBER is set to 20002.
- The default setting for the SOCKETCLOSE attribute is NO. In this case, when a connection is made between a Web service client and CICS, CICS keeps the connection open until the Web service client closes the connection. Alternatively, you can set a value (in seconds) for the SOCKETCLOSE attribute if you want to close a persistent connection after the timeout period is reached.

  We set SOCKETCLOSE to 30 so that connections persist but that an idle connection is timed out after 30 seconds. We also configured WebSphere Application Server and DataPower to persist connections. It is important to enable persistent connections when using SSL because it minimizes the cost of SSL handshaking.

  Recommendation: Use persistent connections where possible. Do not set the SOCKETCLOSE attribute to 0 because this will close the connection after each request.

When it is not possible to use persistent connections (for example, when the Web service client does not support them), it may be possible to reuse SSL session IDs (see “Security considerations” on page 205).

- Setting SSL to Clientauth means that CICS expects to receive a client certificate from the partner system during the SSL handshake.

Note that an equivalent TCPIPSERVICE definition was created for brand XYZ using port 20001.

URIMAP

The URIMAP definition specifies the Web service and pipeline, and sets the specific transaction ID for this service request. The URIMAP is shown in Figure 7-6.

```
OVERTYPE TO MODIFY                                          CICS RELEASE = 0640
CEDA  DEFine Urimap(AcntTABC )
    Urimap         : AcntTABC
    Group          : WSIGOR
  Description ==> URIMAP for brand ABC Account Transfer service
    STatus       ==> Enabled            Enabled | Disabled
    USAge        ==> Pipeline           Server | Client | Pipeline
   UNIVERSAL RESOURCE IDENTIFIER
    SCheme       ==> HTTPS              HTTP | HTTPS
    HOST         ==> abc.pssc.fr
    (Lower Case) ==>
    PAth         ==> /Accounts/AccountTransfer

   ASSOCIATED CICS RESOURCES
    TCpipservice ==> TCPIPABC
    TRansaction  ==> TABC
    PIpieline    ==> PIPEHIGH
    Webservice   ==> AcntTrn
```

Figure 7-6 URIMAP AcntTABC

- Scheme is set to HTTPS so that the service only accepts requests using HTTPS.

Important: When a URIMAP definition with HTTPS as the scheme matches a request that a Web client is making, CICS checks that the inbound port used by the request is using SSL. If SSL is not specified for the port, the request is rejected with a 403 (Forbidden) status code. When the URIMAP definition applies to all inbound ports, this check ensures that a Web client cannot use an unsecured port to access a secured resource. No check is carried out for a URIMAP definition that specifies HTTP as the scheme, so Web clients can use either unsecured or secured (SSL) ports to access these resources.

- Host specifies the host component of the URI to which the URIMAP definition applies abc.pssc.fr.

  Note that an equivalent URIMAP definition was created to match the host URI xyz.pssc.fr for brand XYZ.
- Path specifies the path component of the URI to which the URIMAP definition applies /Account/AccountTransfer.
- The TCPIPSERVICE attribute of a URIMAP defines an inbound port to which this URIMAP definition relates. If this attribute is not specified, the URIMAP definition applies to a request on any inbound port.
- Transaction TABC **(T**ransfer for brand **ABC**) is the name of the transaction ID to be used for the pipeline transaction.
- Pipeline PIPEHIGH specifies the name of the PIPELINE resource definition for the Web service.
- Web service AcntTrn specifies the name of the WEBSERVICE.

PIPELINE

The PIPELINE definition specifies the location of the provider pipeline configuration file that configures the pipeline and determines the header processing program that will be called as part of the pipeline process for high importance services such as account transfers. The PIPELINE is shown in Figure 7-7 on page 191.

```
OVERTYPE TO MODIFY                                          CICS RELEASE = 0640
CEDA  View PIpeline( PIPEHIGH )
 PIpeline       : PIPEHIGH
 Group          : WSIGOR
 Description    : Pipeline for high importance services
 STatus         : Enabled            Enabled ! Disabled
 Configfile     : /group/config/highvalueprovider.xml
 (Mixed Case)   :
                :
```

Figure 7-7 PIPELINE PIPEHIGH

The Configfile specifies the location of the pipeline configuration file highvalueprovider.xml.

WEBSERVICE

The WEBSERVICE definition specifies the name of the Account transfer Web service. The WEBSERVICE is shown in Figure 7-8.

```
OVERTYPE TO MODIFY                                          CICS RELEASE = 0640
CEDA  View Webservice( AcntTrn  )
 Webservice     : AcntTrn
 Group          : WSIGOR
 Description    : HIGH VALUE SERVICE
 Pipeline       : PIPEHIGH
 Validation     : No                 No ! Yes
 WSBind         : /group/wsbind/AccountTransfer.wsbind
 (Mixed Case)   :

 WSDlfile       : /group/wsdl/AccountTransfer.wsdl
 (Mixed Case)   :
                :
```

Figure 7-8 WEBSERVICE AcntTrn

- Pipeline specifies the name of the associated pipeline PIPEHIGH.
- WSBind specifies the location of the bind file AccountTransfer.wsbind.
- WSDlfile specifies the location of the WSDL file AccountTransfer.wsdl.

Pipeline configuration file and header processing program

The service part of the high value provider pipeline configuration file is shown Example 7-2.

Example 7-2 highvalueprovider pipeline configuration file

```
<service>
    <terminal_handler>
      <cics_soap_1.1_handler>
        <headerprogram>
         <program_name>SRSETUID</program_name>
           <namespace>http://docs.oasis-open.org/wss/2004/01/oasis-
200401-wss-wssecurity-secext-1.0.xsd</namespace>
         <localname>Security</localname>
         <mandatory>true</mandatory>
        </headerprogram>
      </cics_soap_1.1_handler>
    </terminal_handler>
  </service>
```

The SRSETUID header processing program extracts the RACF user ID from the Security header and writes it to the DFHWS-USERID container. This then causes a context switch to occur when invoking the wrapper program so that the wrapper program runs with the asserted identity. An example of a header processing program similar to SRSETUID is provided in *Implementing CICS Web Services*, SG24-7206.

SRSETUID also writes an audit record to a user journal.

Important: SRSETUID is added with Mandatory option *True* so that the program is always invoked even if SOAP request does not contain a Security header. SRSETUID issues a SOAP fault if there is no Security header.

Account inquiry service

Here are the set of CICS resource definitions used for processing a Web service request with URI `https://abc.pssc.fr:20002/Accounts/AccountInquiry`.

URIMAP

The URIMAP definition specifies the Web service and pipeline, and sets the specific transaction ID for this service request. The URIMAP is shown in Figure 7-9 on page 193.

```
OVERTYPE TO MODIFY                                          CICS RELEASE = 0640
CEDA  DEFine Urimap(AcntIABC )
    Urimap         : AcntIABC
    Group          : WSIGOR
  Description ==> URIMAP for brand ABC Account Inquiry service
    STatus       ==> Enabled            Enabled | Disabled
    USAge        ==> Pipeline           Server | Client | Pipeline
   UNIVERSAL RESOURCE IDENTIFIER
    SCheme       ==> HTTPS              HTTP | HTTPS
    HOST         ==> abc.pssc.fr
    (Lower Case) ==>
    PAth         ==> /Accounts/AccountInquiry

   ASSOCIATED CICS RESOURCES
    TCpipservice ==> TCPIPABC
    TRansaction  ==> IABC
    PIpieline    ==> PIPESTND
    Webservice   ==> AcntInq
```

Figure 7-9 URIMAP AcntIABC

We now list the AcntIABC URIMAP attributes that are different than those shown for URIMAP AcntTABC in Figure 7-6 on page 189.

- Path specifies the path component of the URI to which the URIMAP definition applies /Account/AccountInquiry.
- Transaction IABC **(I**nquire for brand **ABC**) is the name of the transaction ID to be used for the pipeline transaction.
- Pipeline PIPESTND specifies the name of the PIPELINE resource definition for the Web service.
- Web service AcntInq specifies the name of the WEBSERVICE.

PIPELINE

The PIPELINE definition specifies the location of the provider pipeline configuration file that configures the pipeline and determines the header processing program that will be called as part of the pipeline process for standard services such as Account inquiries. The PIPELINE is shown in Figure 7-10.

```
OVERTYPE TO MODIFY                                          CICS RELEASE = 0640
CEDA  View PIpeline( PIPESTND )
 PIpeline       : PIPESTND
 Group          : WSIGOR
 Description    : Pipeline for standard services
 STatus         : Enabled            Enabled ! Disabled
 Configfile     : /group/config/standardprovider.xml
 (Mixed Case)   :
                :
```

Figure 7-10 PIPELINE PIPESTND

The Configfile specifies the location of the pipeline configuration file standardprovider.xml.

WEBSERVICE

The WEBSERVICE definition specifies the name of the Account inquiry Web service. The WEBSERVICE is shown in Figure 7-11.

```
OVERTYPE TO MODIFY                                          CICS RELEASE = 0640
CEDA  View Webservice( AcntInq )
 Webservice     : AcntTrn
 Group          : WSIGOR
 Description    : STANDARD SERVICE
 Pipeline       : PIPESTND
 Validation     : No                 No ! Yes
 WSBind         : /group/wsbind/AccountInquiry.wsbind
 (Mixed Case)   :

 WSDlfile       : /group/wsdl/AccountInquiry.wsdl
 (Mixed Case)   :
                :
```

Figure 7-11 WEBSERVICE AcntInq

- Pipeline specifies the name of the associated pipeline PIPESTND.
- WSBind specifies the location of the bind file AccountInquiry.wsbind.
- WSDlfile specifies the location of the WSDL file AccountInquiry.wsdl.

Pipeline configuration file and header processing program

The service part of the standard provider pipeline configuration file is shown in Example 7-3.

Example 7-3 Standard provider pipeline configuration file

```
<service>
    <terminal_handler>
      <cics_soap_1.1_handler>
        <headerprogram>
         <program_name>SRSETUIN</program_name>
           <namespace>http://docs.oasis-open.org/wss/2004/01/oasis-
200401-wss-wssecurity-secext-1.0.xsd</namespace>
         <localname>Security</localname>
         <mandatory>false</mandatory>
        </headerprogram>
      </cics_soap_1.1_handler>
    </terminal_handler>
  </service>
```

The header processing program SRSETUIN performs no actions; however, it is required because the incoming SOAP message has a *MustUnderstand* Security header.

The SRSETUIN header processing program is added with the Mandatory option *False* so that the program is not invoked if a SOAP request does not contain a Security header. This permits the standard pipeline to be used for requests that do not have a WS-Security header.

7.2.2 CICS as a service requester

In Figure 7-4 on page 184, we show CICS acting as a service requester. We now look at the CICS definitions that are required to support an outbound service call from CICS for the Account check service (a credit check service provided by an external service provider).

Account check service

Here are the set of CICS resource definitions used for processing an outbound Web service request with the URL `https://ext1.pssc.fr:9137/Accounts/AccountCheck`.

The structure of the URL is as follows:

https	The protocol used to invoke the service.
ext1.pssc.fr	The host name of the external service provider.
9137	The TCP/IP port used by the external service provider.
Accounts	Specifies that the request is for the Accounts application.
AccountCheck	Specifies that the request is for the Account check service.

Note: The URL is defined in the WSDL of the Account check service as the service address location. It is stored in the wsbind file when the CICS Web services utility DFHWS2LS is used to create the language structure and wsbind file for the CICS service requester program.

WEBSERVICE

The WEBSERVICE definition specifies the name of the Account check Web service. The WEBSERVICE is shown in Figure 7-12.

```
OVERTYPE TO MODIFY                                          CICS RELEASE = 0640
CEDA  View Webservice( AcntChk )
 Webservice     : AcntChk
 Group          : WSOGOR
 Description    : Account check Web service
 Pipeline       : PIPEEXT
 Validation     : No                 No ! Yes
 WSBind         : /group/wsbind/AccountCheck.wsbind
 (Mixed Case)   :

 WSDlfile       : /group/wsdl/AccountCheck.wsdl
 (Mixed Case)   :
                :
```

Figure 7-12 WEBSERVICE AcntChk

- Pipeline specifies the name of the associated pipeline PIPEEXT.
- WSBind specifies the location of the bind file AccountCheck.wsbind.
- WSDlfile specifies the location of the WSDL file AccountCheck.wsdl.

PIPELINE

The PIPELINE definition specifies the location of the requester pipeline configuration file that configures the pipeline and determines the header processing program that will be called as part of the pipeline process for outbound service calls. The PIPELINE is shown in Figure 7-13 on page 197.

```
OVERTYPE TO MODIFY                                        CICS RELEASE = 0640
CEDA  View PIpeline( PIPEEXT )
 PIpeline       : PIPEEXT
 Group          : WSOGOR
 Description    : Pipeline for outbound service calls
 STatus         : Enabled            Enabled ! Disabled
 Configfile     : /group/config/requester.xml
 (Mixed Case)   :
                :
```

Figure 7-13 PIPELINE PIPEEXT

The Configfile specifies the location of the pipeline configuration file requester.xml.

Pipeline configuration file and header processing program

The service part of the requester pipeline configuration file is shown in Example 7-4.

Example 7-4 Requester pipeline configuration file

```
<service>
    <terminal_handler>
      <cics_soap_1.1_handler>
        <headerprogram>
         <program_name>SRSETUIO</program_name>
           <namespace>http://docs.oasis-open.org/wss/2004/01/oasis-
200401-wss-wssecurity-secext-1.0.xsd</namespace>
          <localname>Security</localname>
          <mandatory>true</mandatory>
        </headerprogram>
      </cics_soap_1.1_handler>
    </terminal_handler>
  </service>
```

The SRSETUID header processing program obtains the user ID from the DFHWS-USERID container, creates a Security header, and puts the header into the DFHHEADER container. CICS then attaches the header to the SOAP message that it sends to the service provider.

An example of a header processing program similar to SRSETUIO is provided in *Implementing CICS Web Services*, SG24-7206.

7.3 Building a high availability configuration

Downtime is costly. The cost of downtime is so great that many of today's enterprises can no longer afford planned or unplanned outages. Even beyond the financial aspects, downtime can also affect key areas of customer loyalty, market competitiveness, and regulatory compliance. A robust configuration designed for high availability is therefore a necessity, especially in financial systems.

In this section, we outline the main availability capabilities of our configuration.

7.3.1 CICSplex configuration

For the CICS service provider scenario (Figure 7-3 on page 183) an *Inbound Gateway Owning Region* (IGOR) hosts the wrapper program. The wrapper program then links to a remote business logic program running in an AOR.

The Web services related resource definitions described in 7.2.1, “CICS as a service provider” on page 187 are all installed on the IGORs.

Note: In this project example, an Inbound Gateway Owning Region (IGOR) is like a TOR, but it is specific to the processing of Web service requests.

Figure 7-14 on page 199 shows a wrapper program acting as a service provider application. The program link to the business logic program is workload managed across a set of brand specific AORs using CICSPlex SM workload management. A brand AOR runs applications for a particular brand (brand ABC or XYZ).

The IGORs that host the wrapper program listen on a shared port (see 4.2.1, “Port sharing” on page 81 for a description of TCP/IP Port sharing).

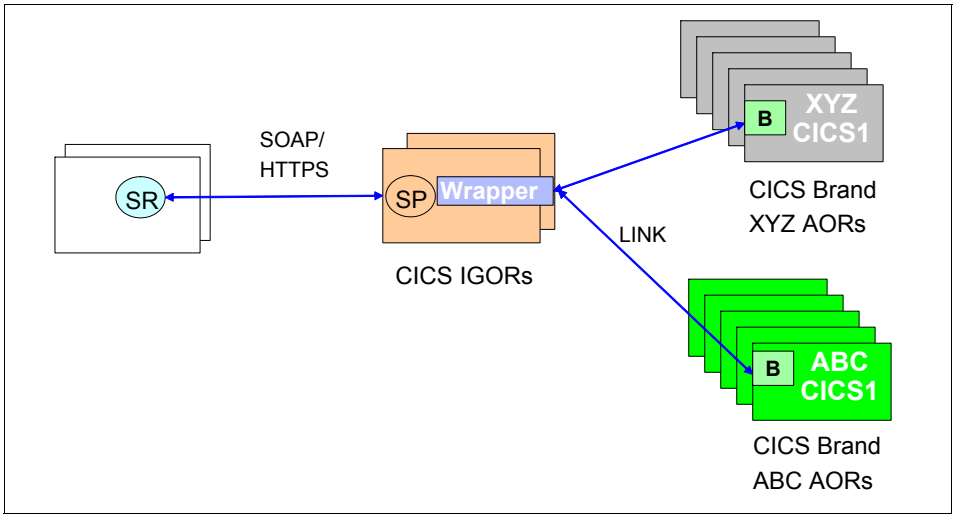


Figure 7-14 Inbound Gateway Owning Region

For the CICS service requester scenario (Figure 7-4 on page 184), an *Outbound Gateway Owning Region* (OGOR) hosts the wrapper program. Business logic programs in brand AORs do not invoke Web services directly; instead, they link to wrapper programs in OGORs, which make outbound service calls using the EXEC CICS INVOKE WEBSERVICE API. The program links are workload managed across a set of OGORs using CICSPlex SM workload management. Figure 7-15 shows a wrapper program acting as a service requester application.

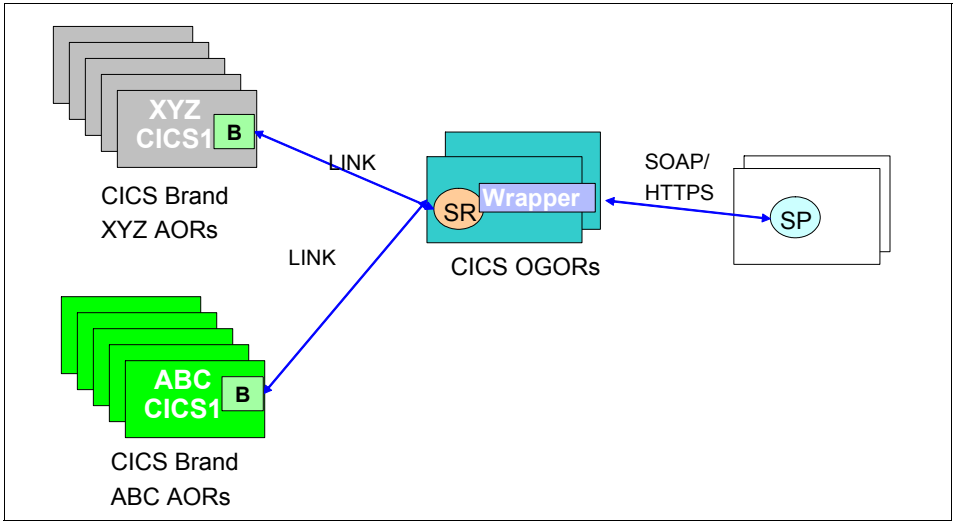


Figure 7-15 Outbound Gateway Owning Region

The Web services related resource definitions described in 7.2.2, “CICS as a service requester” on page 195 are all installed on the OGORs.

7.3.2 CICSPlex SM definitions

We used the CICSPLex SM Web User Interface (WUI) to define our CICSPlex, including CICS system definitions, CICS connection definitions, and workload management definitions.

CICS system definitions

Figure 7-16 shows the CICS systems defined for our CICSplex.

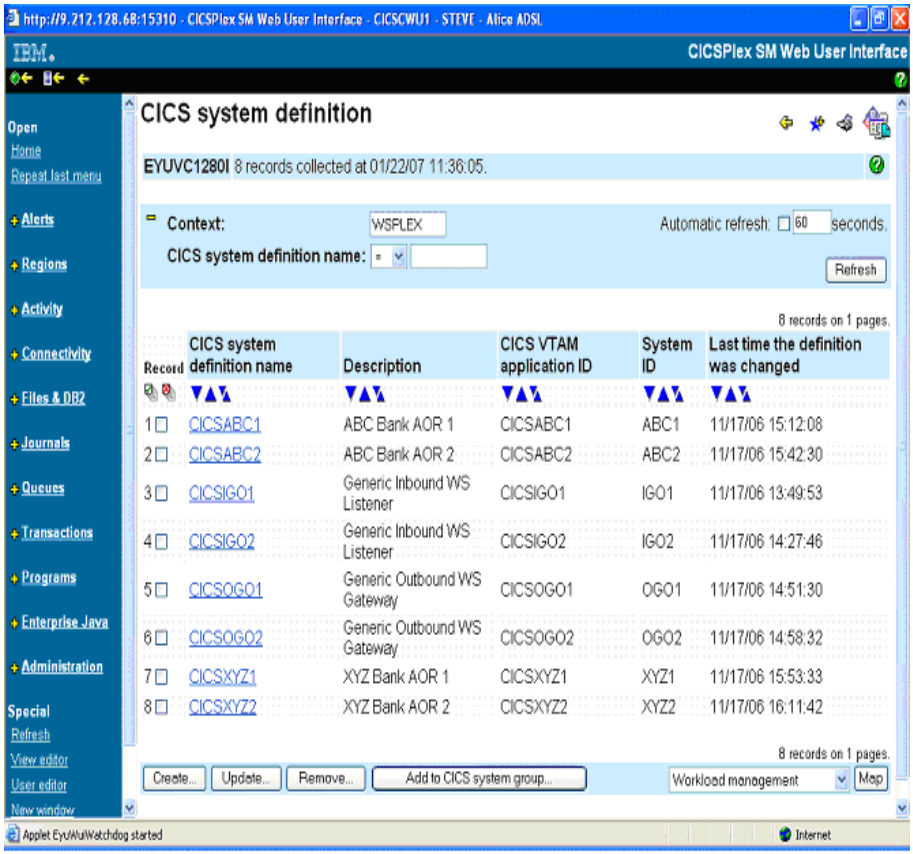


Figure 7-16 CICSplex SM WUI - CICS system definitions

CICS connection definitions

We used the SYSLINK facility provided with BAS to define the MRO connections between the CICS systems within the WSPLEX CICSplex.

Workload management definitions

When a service request arrives at an IGOR, the URI is associated to a URIMAP and the URIMAP specifies the transaction ID to be used for the pipeline transaction and wrapper program (see “URIMAP” on page 192). The different transactions used for our workload are shown in Table 7-2.

Table 7-2 Pipeline transaction IDs

Service	Brand	Transaction ID
Transfer	ABC	TABC
Inquire	ABC	IABC
Transfer	XYZ	TXYZ
Inquire	XYZ	IXYZ

When the wrapper program running in an IGOR links to a business logic program, the program link is dynamically routed to a set of brand AORs using CICSPlex SM workload management. The remote program definitions for business logic programs in the IGORs include a Transid so that the same transaction IDs shown in Table 7-2 are also used in the AORs for the mirror transactions.

Figure 7-17 shows a map of the Workload Management (WLM) specification that is associated with the IGORs.

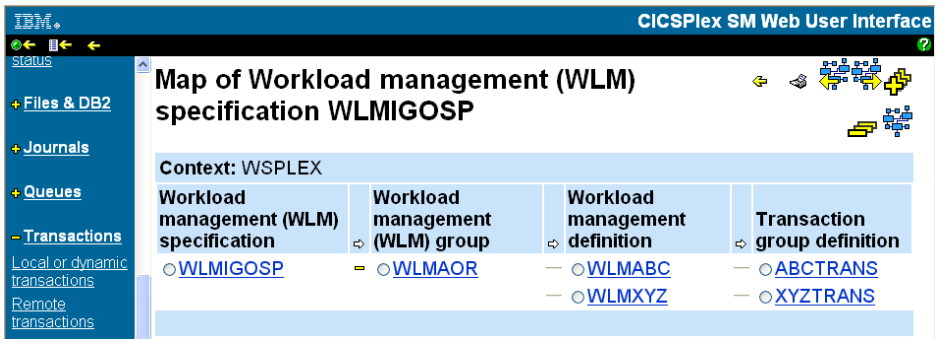


Figure 7-17 CICSplex SM WUI - WLM specification WLMIGOSP

- WLMIGOSP is the name of the WLM specification.

  This specification is used in the IGORs to route the dynamic program links to business logic programs in the AORs.

- WLMAOR is the workload management group used for CICS service provider applications.

- The WLMABC and WLMXYZ workload definitions are associated with the WLMAOR workload management group.

  – WLMABC uses the transaction group ABCTRANS, which contains the brand ABC transactions TABC and IABC. It routes these transactions to the brand ABC AORs CICSABC1 and CICSABC2.

  – WLMABC uses the transaction group XYZTRANS, which contains the brand XYZ transactions TXYZ and IXYZ. It routes these transactions to the brand XYZ AORs CICSXYZ1 and CICSXYZ2.

See 5.6, “Workload management” on page 147 for information about how to make workload management definitions using the CICSPlex SM WUI Workload manager administration views.

Figure 7-18 on page 203 shows a map of the Workload Management (WLM) specification that is associated with the AORs.

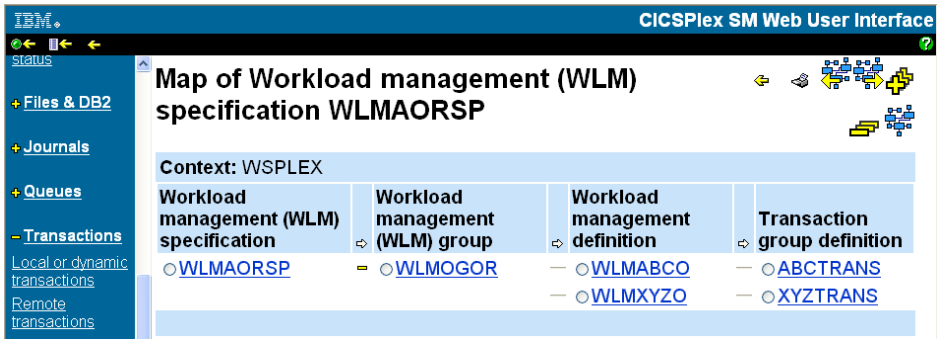


Figure 7-18 CICSplex SM WUI - WLM specification WLMAORSP

- WLMAORSP is the name of the WLM specification.

  This specification is used in the AORs to route the dynamic program links to wrapper programs in the OGORs.

- WLMOGOR is the workload management group used for CICS service requester programs (the wrapper programs that make outbound Web service calls).

- The WLMABCO and WLMXYOZ workload definitions are associated with the WLMOGOR workload management group.

  – WLMABCO uses the transaction group ABCTRANS, which contains the brand ABC transactions TABC and IABC. It routes these transactions to the OGORs CICSOGO1 and CICSOGO2.

  – WLMABCO uses the transaction group XYZTRANS, which contains the brand XYZ transactions TXYZ and IXYZ. It routes these transactions to the OGORs CICSOGO1 and CICSOGO2

Note: The two workload definitions WLMABCO and WLMXYOZ route transactions to the same OGORs. Two workload definitions are used; however, because the same transaction IDs are workload managed in the WLMABC and WLMXYZ workload definitions, it is not possible to include a transaction in more than one transaction group.

7.3.3 Parallel sysplex configuration

When deploying CICS Web services in a parallel sysplex, you can take advantage of the z/OS specific workload management capabilities, including Sysplex Distributor, MVS Workload Manager (WLM), and shared MVS logstreams. Figure 7-19 shows the parallel sysplex configuration tested for the CICS service provider scenario.

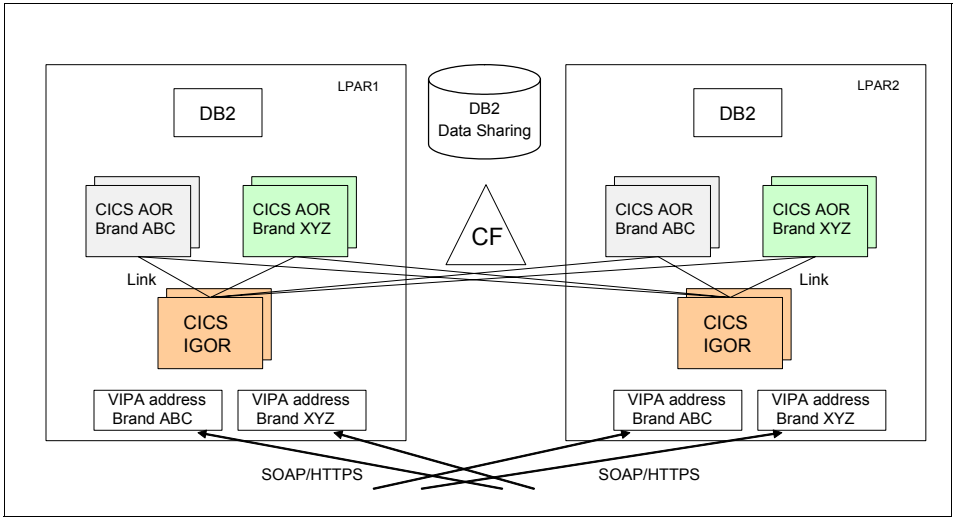


Figure 7-19 Parallel sysplex configuration

- Sysplex Distributor is used for workload management of TCP/IP connections across two LPARs (LPAR1 and LPAR2).

  A different dynamic VIPA address and port combination is used for each brand or subsidiary of the financial group (for example, retail bank or insurance company). This establishes a virtual separation of Web services traffic across the different brands.

  See 5.2, “TCP/IP definitions” on page 107 for information about how to configure Sysplex Distributor.

- Multiple IGORs on each LPAR listen on a shared TCP/IP port. See 5.2, “TCP/IP definitions” on page 107 for information about how to configure TCP/IP port sharing.
- Program link requests are dynamically routed to cloned brand AORs using CICSPLex SM.

- A user logstream is shared in the coupling facility so that all IGORs write audit records to the same journal (audit records are written for each high value service request). The audit records are written by the header processing program SRSETUID.
- CICS business logic programs running in the AORs share access to business data using DB2 data sharing.

Security considerations

It is important to enable persistent TCP/IP connections when using SSL because it minimizes the cost of SSL handshaking. We set the SOCKETCLOSE attribute of the IGOR TCPIPSERVICEs to 30 so that connections persist and idle connections are timed out after 30 seconds.

When TCP/IP connections do not persist, or if the connection has timed out, it is still possible to avoid the impact of a full SSL handshake by reusing an SSL session ID. Session IDs are tokens that represent a secure connection between CICS and an SSL client. The session ID is created and exchanged between the SSL client and CICS during the SSL handshake. If a match is found, then an SSL session can be established without the impact of validating SSL certificates.

The CICS SIT parameters SSLDELAY and SSLCACHE are used to control how CICS manages SSL session IDs.

- SSLDELAY={600 | *number*}

  Specifies the length of time in seconds for which CICS retains session IDs for secure socket connections in the SSL cache. While the session ID is retained by CICS within the SSLDELAY period, CICS will re-establish an SSL connection with a client by using only a partial handshake. We used the default value of 600 seconds.

- SSLCACHE={CICS | SYSPLEX}

  Specifies whether CICS should use the local SSL cache in the CICS region, or share the cache across multiple CICS regions by using the sysplex session cache support provided by System SSL.

With SSLCACHE=CICS, a client who successfully connects to CICS region 1 on z/OS system 1 and then connects to CICS region 2 on z/OS system 2 must go through a full SSL handshake in both cases; this is because CICS stores the SSL session ID in a cache that is local to the CICS address space.

With SSLCACHE=SYSPLEX, an SSL session established with a CICS region on one system in the sysplex can be resumed using a CICS region on another system in the sysplex as long as the SSL client presents the session identifier obtained for the first session when initiating the second session. CICS uses the sysplex session cache support provided by the System SSL started task (GSKSRVR), an optional component of System SSL.

We set SSLCACHE to SYSPLEX.

Tip: Refer to the PK46069 and IY99898 APARs if you intend to use SSL session ID reuse between WebSphere Application Server and CICS.

7.4 Performance and scalability

In this section, we provide a summary of some of the performance results that we achieved. We also discuss some of the factors that we found to have a significant impact on performance.

We defined a set of tests to validate the scalability and performance of the solution:

- Gradually increasing the number of simulated Web service invocations to test linear scalability
- Investigating the performance impact of varying SOAP message lengths
- Measuring the performance cost of different Web service security mechanisms

7.4.1 Linear scalability

Linear scalability implies that the CPU cost and response time of each service request remain relatively constant as the service hit rate increases.

Figure 7-20 on page 207 shows the CPU consumption of the Account inquiry service for different service hit rates. CPU consumption is measured by the number of milliseconds of processor (CP) time taken by the IGOR to process a single request.

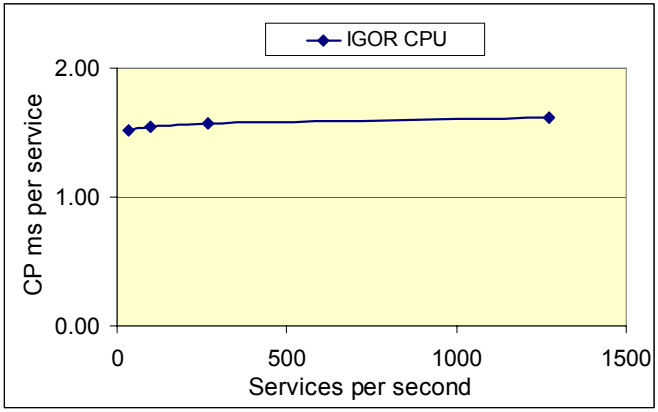


Figure 7-20 Account inquiry service CPU cost

Figure 7-20 shows that the milliseconds of CPU required to process an account inquiry request is relatively constant, demonstrating linear scalability. The response time also demonstrated linear scalability by remaining constant (approximately 200 ms) for tests at different service rates.

7.4.2 SOAP message length

We performed tests to evaluate the effect of message size on the performance of the Account inquiry Web service. We varied the message size by increasing the number of XML elements contained in the response message returned by CICS.

Figure 7-21 shows the effect of message size on CPU consumption for the Account inquiry service.

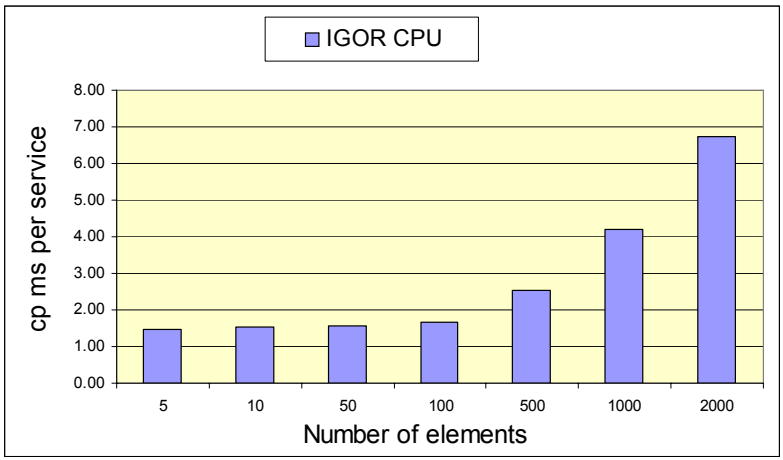


Figure 7-21 SOAP message length impact on CPU cost

Figure 7-21 shows that the milliseconds of CPU required to process an account inquiry request is dependent on the number of elements returned in the response message. In our test, a SOAP message of 50 elements contains 1 KB of business data, but the message itself has a length of 2.6 KB because of the XML tags. The longest message is 400 times larger than the smallest message, but the increase in CPU consumption is less than five fold.

Figure 7-22 shows the effect of message size on response time for the account inquiry service.

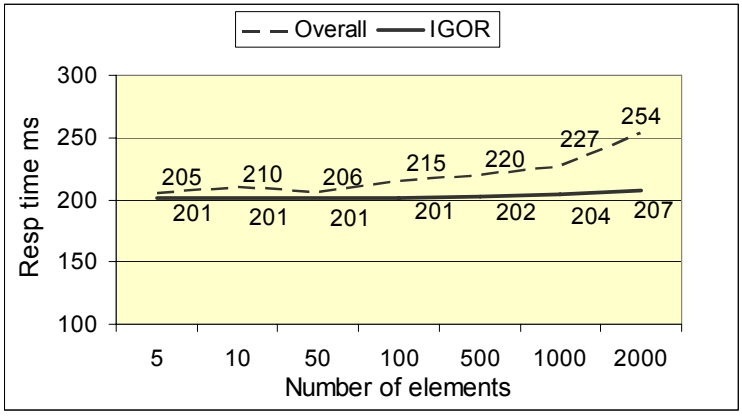


Figure 7-22 SOAP message length impact on response time

Figure 7-22 on page 208 shows the IGOR response time for account inquiry services with varying response message sizes:

- The IGOR response time is the average response time recorded by the RMF™ component of z/OS for the CICS service provider transaction. The response time varies from 201 milliseconds for the smallest message length, to 207 milliseconds for the largest message length. Note that for this test that a 200 ms delay is specifically programmed into the target business logic program, in order to simulate the average response time for existing programs.
- The overall response time is the response time measured in the workload simulation tool, so this includes the response time of the CICS service provider application and the service requester application (a J2EE application running in WebSphere Application Server).

We see that the IGOR response time remains relatively constant while the overall response time increases more significantly for the longer SOAP messages. This is because CICS is building the SOAP message response (no additional XML parsing is required) while WebSphere Application Server is parsing the response message.

Note: The complexity of the SOAP message also has an impact on performance.

For more information about the performance considerations associated with SOAP message size and complexity, refer to *SOAP Message Size Performance Considerations*, REDP-4344.

7.4.3 Security cost

We performed a test to evaluate the effect of the different security models. Figure 7-23 compares the CPU consumption for the two services Account inquiry and Account transfer.

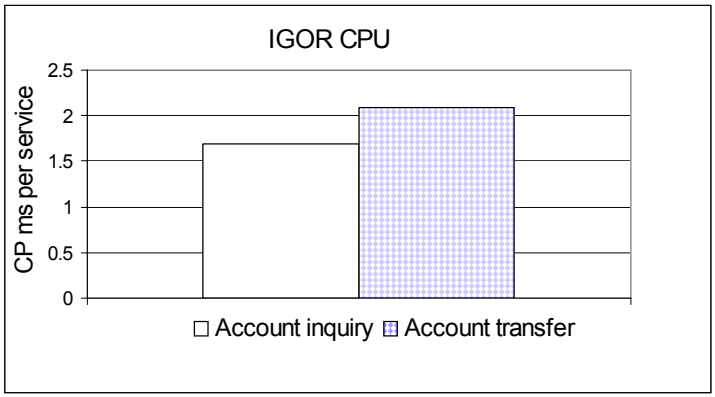


Figure 7-23 Security model impact on CPU cost

Figure 7-23 shows that the Account transfer service CPU cost is approximately 20% greater than the Account inquiry service CPU cost. This is due to the impact of processing the Security header, in particular, the additional cost of the security context switching (see 7.1.6, “Security model” on page 185). For an Account transfer request, the content of the DFHWS-USERID container is changed, which causes a new task to be initiated.

7.5 Operations and monitoring tools

The main operations and monitoring requirements of the solution are as follows:

- Operations capability to monitor a CICS Web services workload
- Ability to monitor against the set of predefined service performance goals
- To prevent problems with one brand impacting the service level of other brands
- Ability to identify a problem when it occurs and to identify the location and root cause of the problem

The following tools are used to operate and monitor the CICS Web services infrastructure:

- The CICSPlex SM WUI is the principal tool for overseeing the Web services workload (see 7.6.1, “Normal workload processing” on page 212), and identifies problems such as CICS region failures (see 7.6.2, “CICS region failures” on page 218) and operations, such as quiescing a CICS region.
- IBM Tivoli OMEGAMON XE for CICS is used for tracking against service response time goals (see 7.6.3, “Service level analysis” on page 220).
- IBM Tivoli Composite Application Manager for SOA (ITCAM for SOA) is used for monitoring Web services across different runtime environments, including CICS and WebSphere DataPower (see 7.6.4, “Message length analysis” on page 225).

Figure 7-24 shows how the Tivoli OMEGAMON XE for CICS and Tivoli ITCAM for SOA tools were installed in our environment. Both tools were used to graphically display information in a single integrated console (the Tivoli Enterprise Portal).

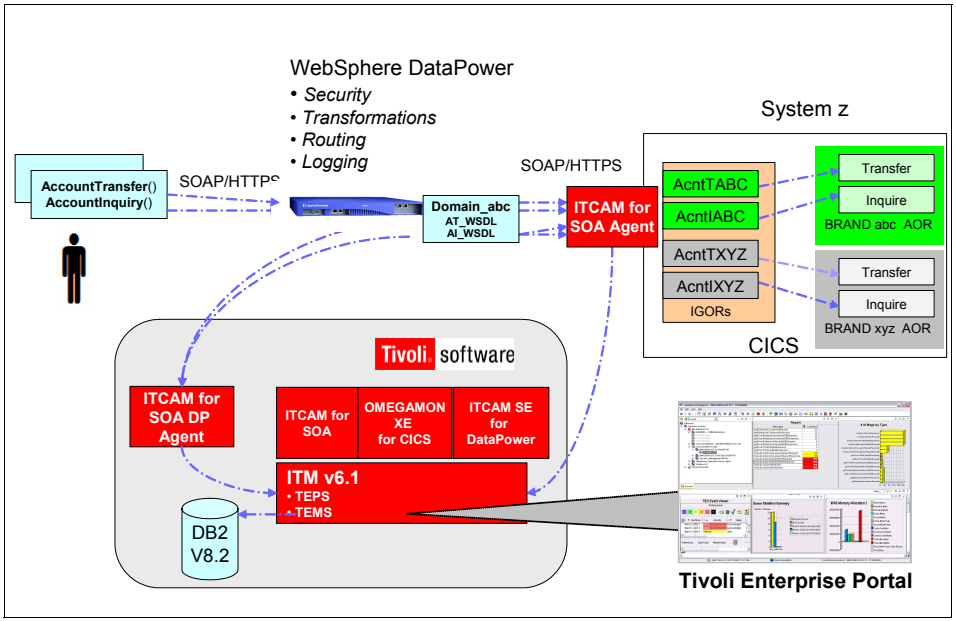


Figure 7-24 Tivoli Monitoring configuration

To monitor all the service platforms, ITCAM for SOA employs (and ships with) a series of agents, such as the CICS agent, which is implemented as a message handler, and the DataPower agent, which resides on a separate machine and queries the DataPower appliance using the DataPower SOAP interface.

7.6 Workload management and monitoring scenarios

In this section, we review some of the tests that were performed in order to validate how the CICS Web service infrastructure meets the workload management, availability, and monitoring non-functional requirements. During the test scenarios, we used different operations and management tools.

7.6.1 Normal workload processing

We used a workload simulation tool to simulate a CICS Web service workload. The workload consisted of Account inquiry and Account transfer service requests for both brands ABC and XYZ. The requests are received by the CICS IGORs and program links are dynamically routed to the brand AORs.

For a subset of the inbound requests, the CICS service provider application needs to call the Account check Web service, which is provided by an external service provider. For such requests, the business logic program running in a brand AOR calls a wrapper program running an OGOR to invoke the external Web service.

The following set of screen captures show how we used the CICSPlex SM WUI views to monitor the CICS Web services workload.

Figure 7-25 shows the TCPIPSERVICES TCPIPABC and TCPIPXYZ for the two IGORs CICSIGO1 and CICSIGO2. It shows active TCP/IP connections for both TCPIPSERVICES across both IGORs.

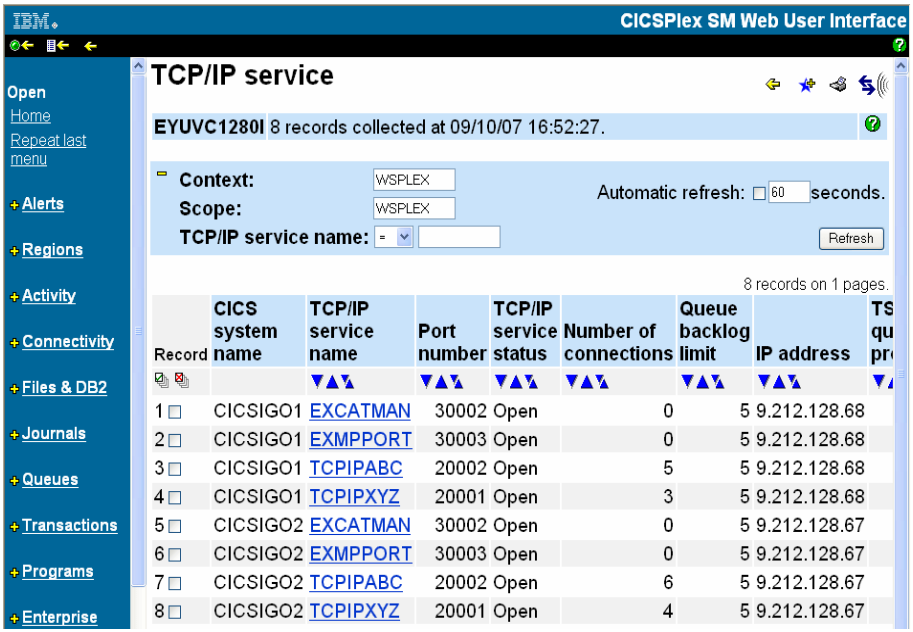


Figure 7-25 CICSPlex SM WUI TCPIPSERVICE view - normal workload

Figure 7-26 shows the number of times that the AcntInq and AcntTrn Web services have been used on the IGORs and the number of times that the AcntChk Web service has been used on the OGORs.

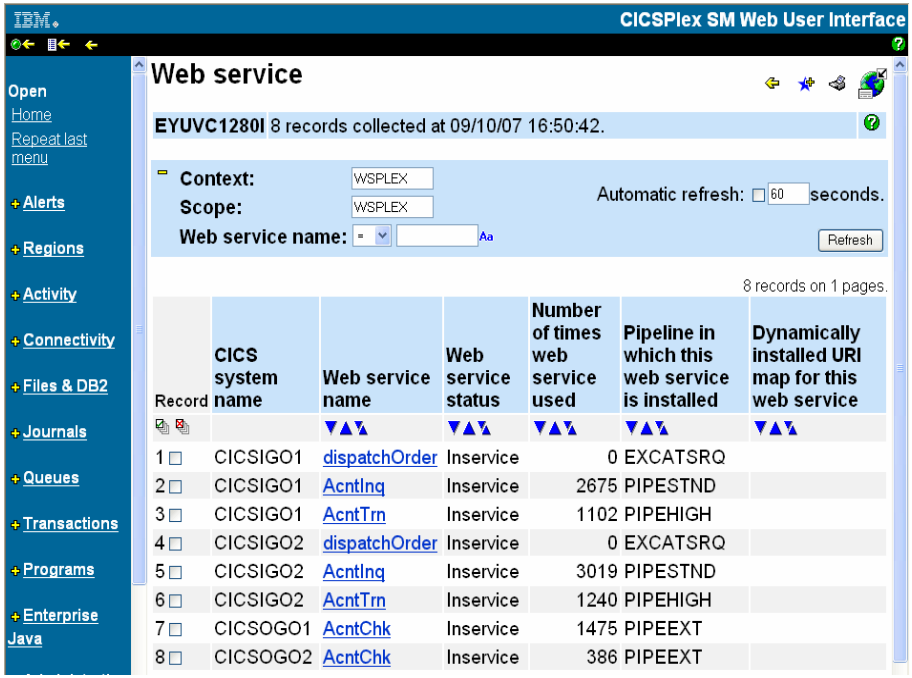


Figure 7-26 CICSPlex SM WUI Web service view - normal workload

Figure 7-27 shows the number of times that the PIPEHIGH and PIPESTND pipelines have been used on the IGORs and the number of times that the PIPEEXT pipeline has been used on the OGORs.

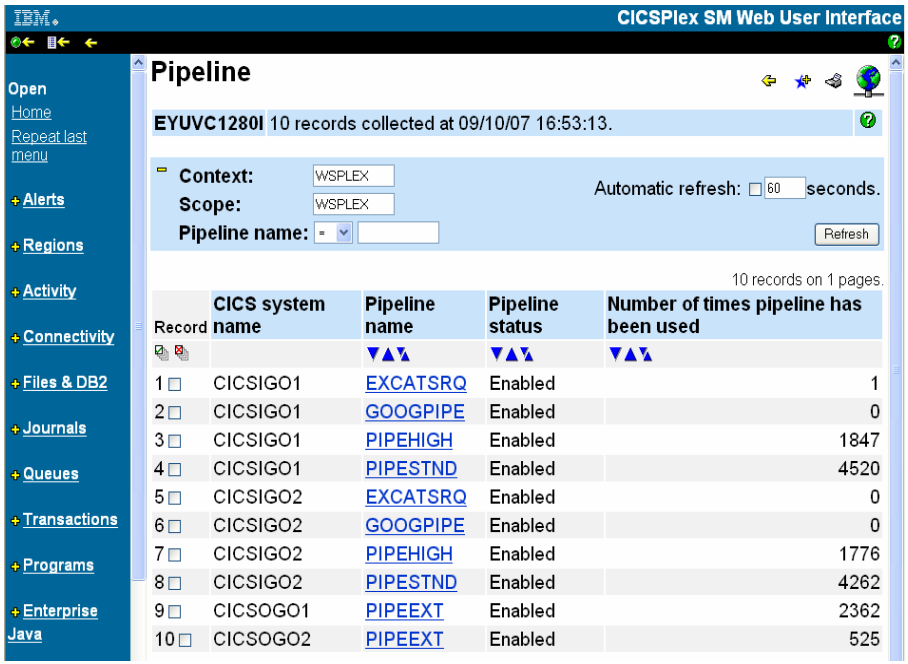


Figure 7-27 CICSPlex SM WUI Pipeline view - normal workload

When a service request arrives at an IGOR, the URIMAP is used to associate a transaction ID with the request (see Table 7-2 on page 201). This means that we can monitor the processing of specific types of service across our CICSplex using the Transactions view of the WUI; for example, Figure 7-28 shows the number of times the brand ABC Account transfer transaction TABC has been used on the different CICS systems within the CICSplex.

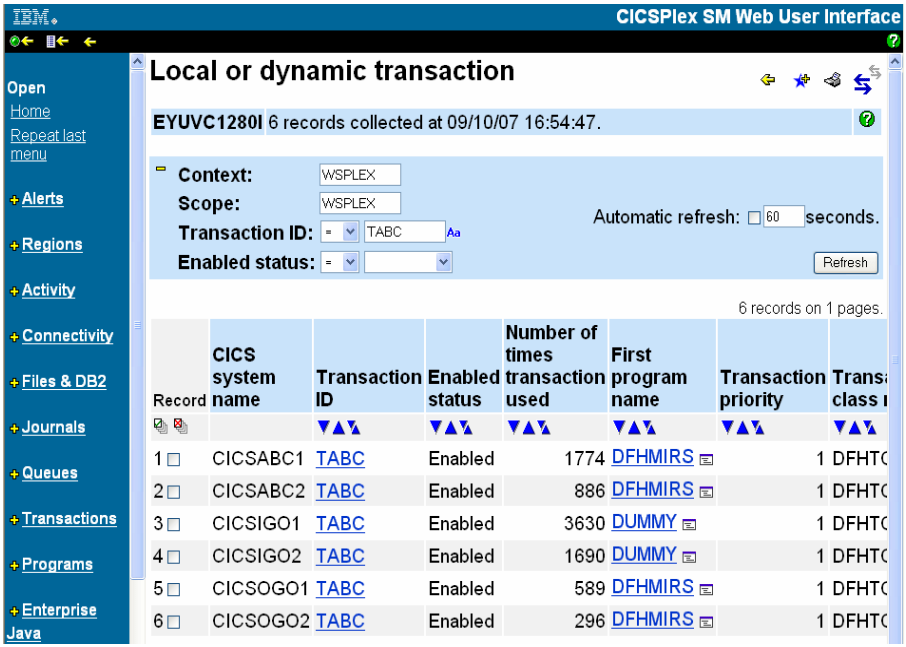


Figure 7-28 CICSPlex SM WUI Transactions view - normal workload

We can also use the WUI to view active brand ABC Account transfer requests using the Active task view (Figure 7-29 on page 217).

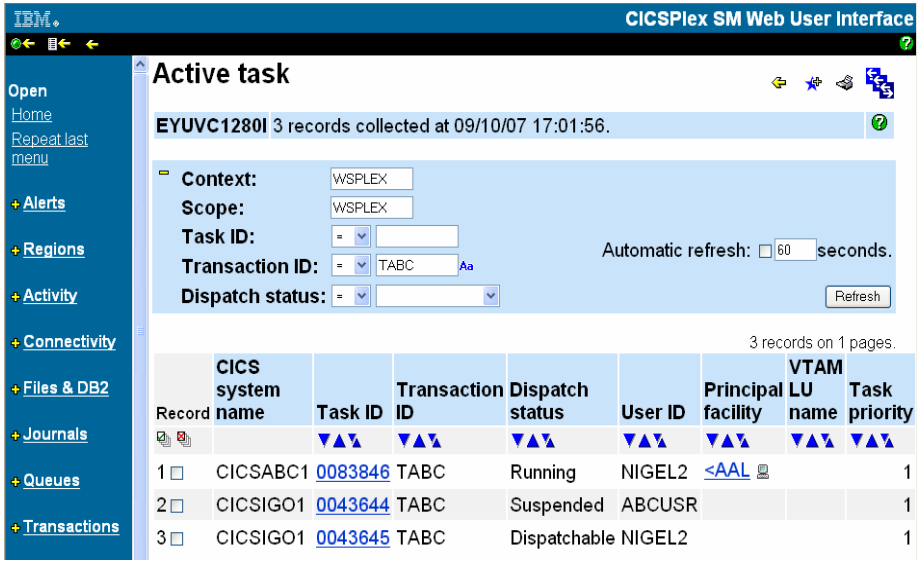


Figure 7-29 CICSPlex SM WUI Active task view - normal workload

Figure 7-29 shows three active tasks which are all associated with the processing of a single Account transfer request:

- Task 43644 running in the IGOR CICSIGO1 is the initial pipeline alias task. This task runs with user ID ABCUSER, which is the user ID mapped from the X.509 certificate used for the client authenticated SSL connection.
- Task 43645, also running in CICSIGO1, is the task used for the wrapper program processing. The wrapper program executes in a new task because the SRSETUID header processing program extracts the RACF user ID from the Security header in the SOAP message and writes it to the DFHWS-USERID container. See Example 7-1 on page 185 for an example of a Security header.

  Important: Any change to the contents of the DFHWS-USERID or DFHWS-TRANID containers causes a new task to be initiated. The original pipeline alias task is suspended until this second task completes.

  Task 43645 runs with user ID NIGEL2, which is the user ID contained in the Security header (the service requester's identity).
- The wrapper program, which runs in the IGOR links to the business logic program. The program link is dynamically routed by CICSPlex SM to an AOR. Task 83846 is the mirror task running in the AOR CICSABC1.

7.6.2 CICS region failures

While the Web services workload was running, we tested the impact of an IGOR failure and an AOR failure.

CICS IGOR failure

In the event of a CICS IGOR failure, active TCP/IP connections to the failed IGOR are lost and Sysplex Distributor redirects new connection requests to the other active IGORs.

Figure 7-30 is a CICSPlex SM WUI TCPIPSERVICE view after CICSIGO1 has been cancelled. It shows that active TCP/IP connections exist only for the TCPIPSERVICES TCPIPABC and TCPIPXYZ on CICSIGO2.

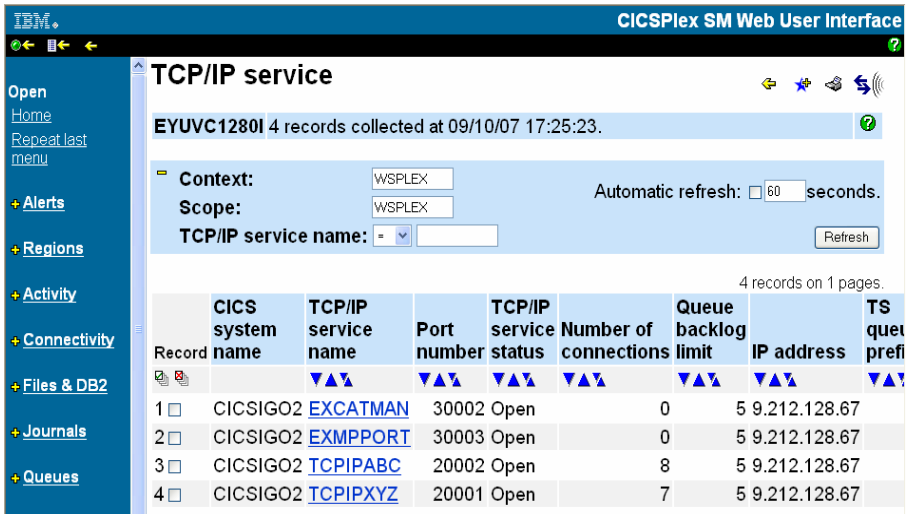


Figure 7-30 CICSPlex SM WUI TCPIPSERVICE view - IGOR failure

The workload simulation tool showed that there was a brief fall in service hit rate, and corresponding increase in service response time, at the time when CICSIGO1 was cancelled. The service hit rate then returned to the same level as before the failure (approximately 30 services per second).

Figure 7-31 on page 219 shows that the AcntInq and AcntTrn Web services are being used only on CICSIGO2.

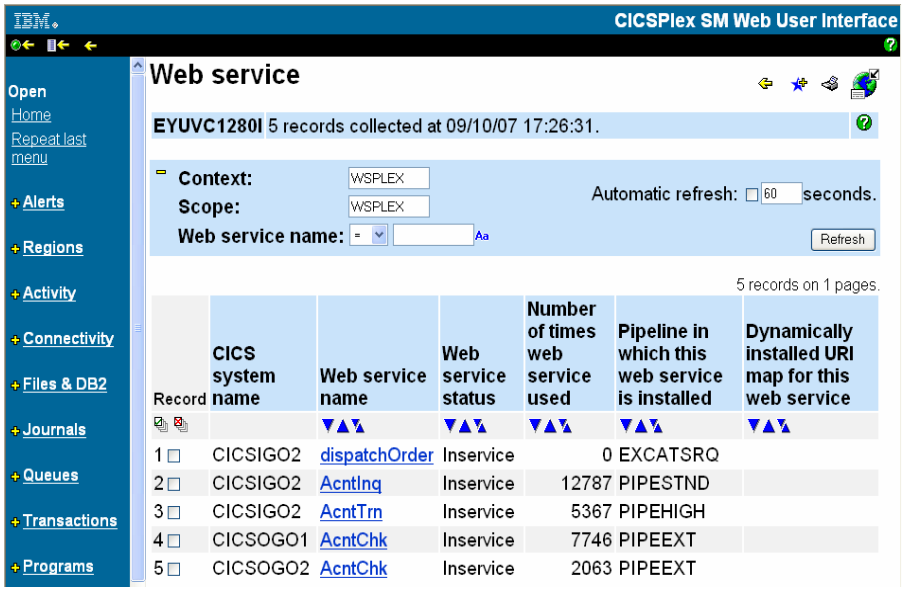


Figure 7-31 CICSPlex SM WUI Web service view - IGOR failure

CICS AOR failure

In the event of a CICS AOR failure, the CICSPlex SM workload balancing routes all program link requests to the remaining active AORs.

Figure 7-32 is a CICSPlex SM WUI Transactions view after CICSABC2 has been cancelled. It shows that the brand ABC Account transfer transaction TABC is being run on CICSABC1.

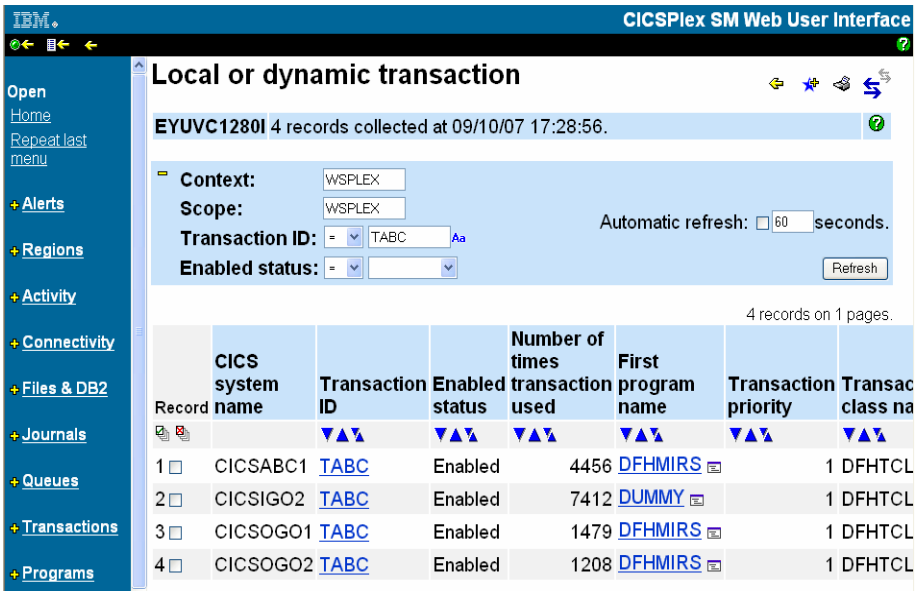


Figure 7-32 CICSPlex SM WUI Transactions view - AOR failure

The CICS region failure tests help to validate that the high availability CICS Web services configuration is able to withstand region failures without significant impact on service availability.

7.6.3 Service level analysis

A requirement of the CICS Web services infrastructure is that a problem associated with a single brand should not impact the availability and performance of work associated with other brands.

In this scenario, we discover that if there is a problem in one brand's AOR that causes the IGOR tasks associated with that brand to be suspended for long periods of time, then the other brand's workload will be unaffected (that is, there is no "sympathy sickness" effect).

We used the monitoring facilities provided by Tivoli Omegamon XE for CICS to demonstrate that a problem in a single CICS region associated with brand XYZ does not affect the brand ABC workload running in the same IGORs.

Figure 7-33 is Service Level analysis view of Tivoli OMEGAMON XE for CICS before the problem is injected into the workload.

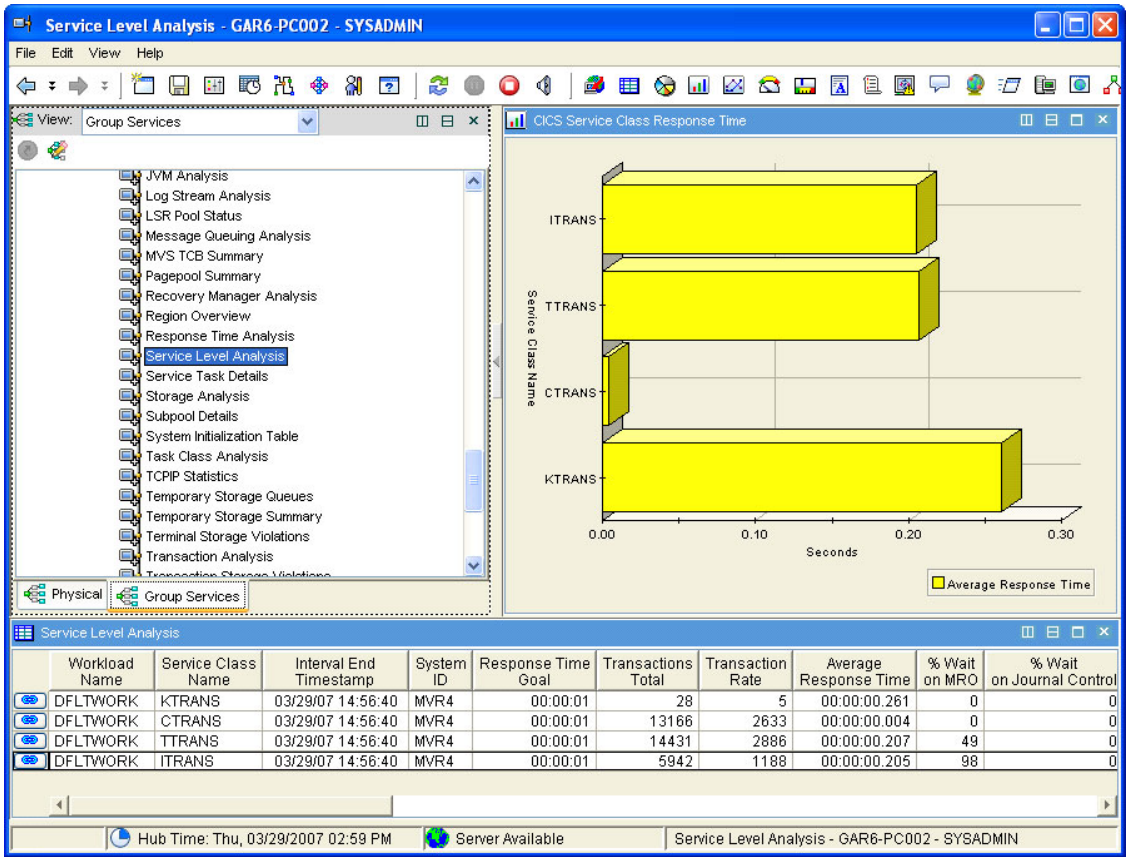


Figure 7-33 OMEGAMON XE for CICS - Service Level Analysis view

Figure 7-33 shows how well the current workload is doing in terms of meeting the service response time goal of one second, which has been set in Tivoli OMEGAMON XE for CICS. This is the same goal that we set for our transactions using MVS WLM.

This default view divides the workload up according to the first letter of the CICS transaction ID under which that work is being executed, so all transactions beginning with the character 'I' are classified under the ITRANS service class and transactions beginning with 'T' are classified under the TTRANS service class. For the purposes of this test, we ran a workload which consisted of brand ABC Account inquiry services (transaction IABC) and brand XYZ Account transfer services (transaction IXYZ).

In Figure 7-33 on page 221, we are interested in the TTRANS and ITRANS service classes. We can see that for the moment all is well with the average response time of around 200ms for both the ITRANS service class (the IABC transaction is in this service class) and the TTRANS service class (the TXYZ transaction is in this service class). This response time is within the defined response time goal of one second. We can also see that for both IABC and TXYZ, there is some MRO wait time, reflecting the fact that there is a program link to an AOR during execution of the transactions.

In Figure 7-34 on page 223, we see the Tivoli OMEGAMON XE for CICS URIMAP Analysis view, giving us an insight into what Web services requests are arriving in our CICSplex.

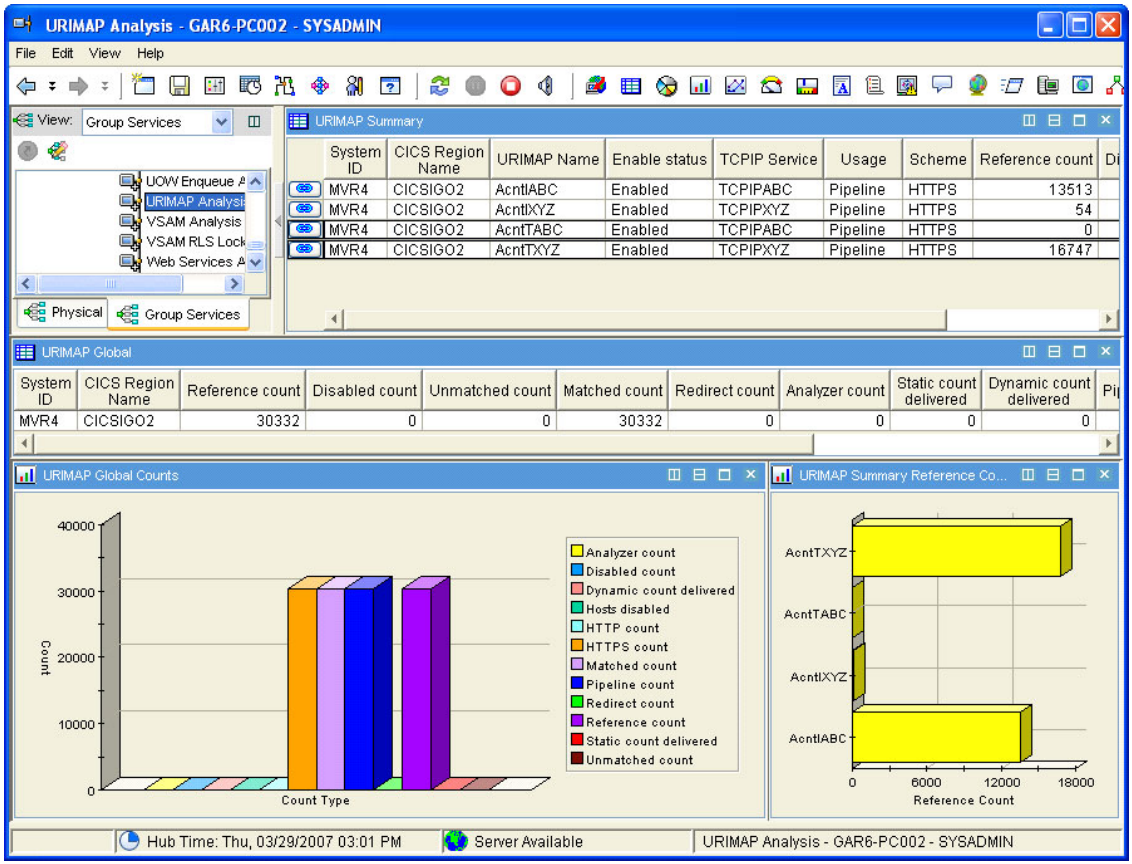


Figure 7-34 OMEGAMON XE for CICS - URIMAP Analysis view

Figure 7-34 shows that the requests arriving in the IGOR are all HTTPS rather than HTTP, that they are all Web service calls (the Pipeline bar is the same height), and that all incoming requests have been successfully matched to an installed URIMAP. We see also that of the four URIMAPs defined, the only two that are being used are the AcntTXYZ and AcntIABC URIMAPs.

At this point, we modified the target business logic program for the brand XYZ Account transfer service to delay for 20 seconds. We observed the effect of this change using the Service Level analysis view of Tivoli OMEGAMON XE for CICS. Figure 7-35 shows that the Service Level view following the change.

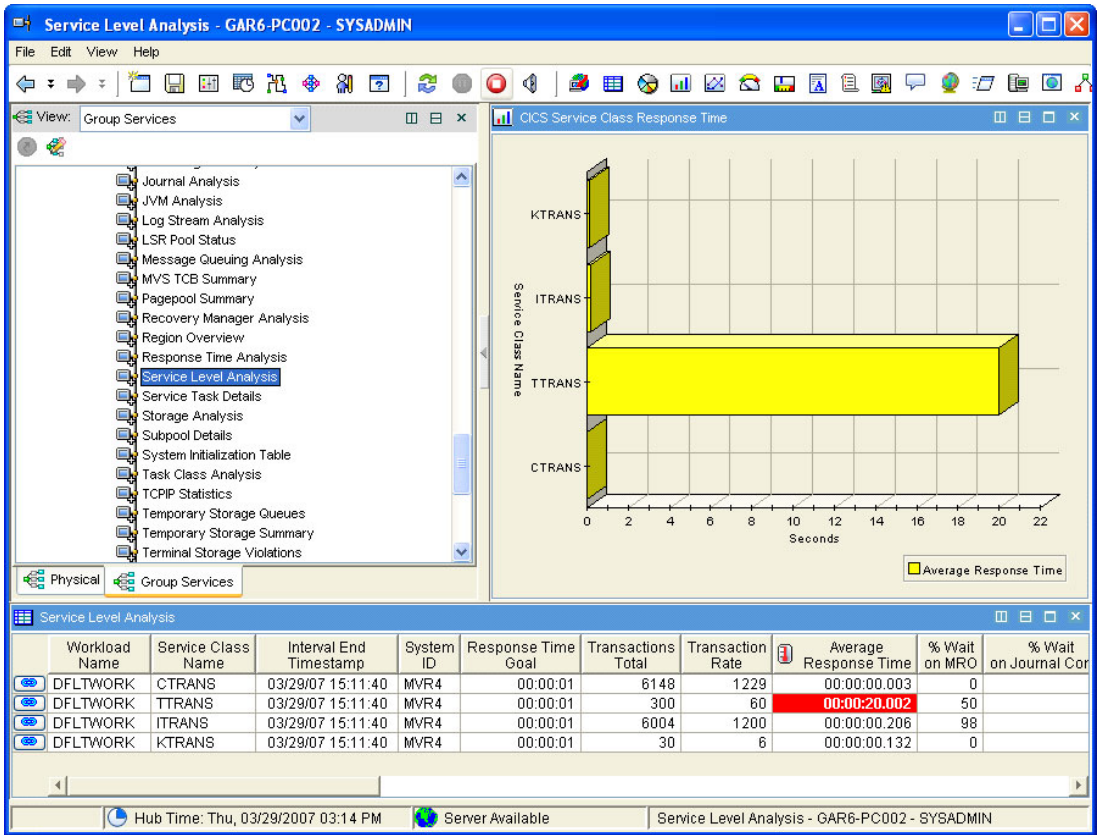


Figure 7-35 OMEGAMON XE for CICS - Service Level Analysis view

Figure 7-35 shows that the average response time of the TXYZ transactions is now 20 seconds, and because it is failing to meet the Response Time Goal, this average response time is highlighted. However, the IABC transaction continues to meet its response time goal of one second, with an average response time of 206 milliseconds. The deterioration in performance of the XYZ brand workload has not affected the performance of the ABC brand workload.

Note that this test is conducted with a constant workload. If we imagine that more and more brand XYZ users attempt to send service requests, it is likely that the brand XYZ response time delay would stall the IGORs. This problem could be avoided by grouping brand specific transaction IDs in a CICS transaction class,

thus restricting the number of concurrent brand specific transactions that can run in an IGOR. In this case, if there is a problem in an AOR that causes tasks to be delayed, the IGOR will not be swamped with new tasks for that brand.

7.6.4 Message length analysis

One of the factors that determines the performance of a Web service, particularly the CPU cost, is the length of SOAP message that the service provider runtime needs to parse. This makes it important to be able to monitor SOAP message length for the different operations of a Web service.

In this scenario, we show how ITCAM for SOA can provide useful information such as message length and service response times.

Figure 7-36 shows the Message Summary view as reported by the CICS agent of the ITCAM for SOA.

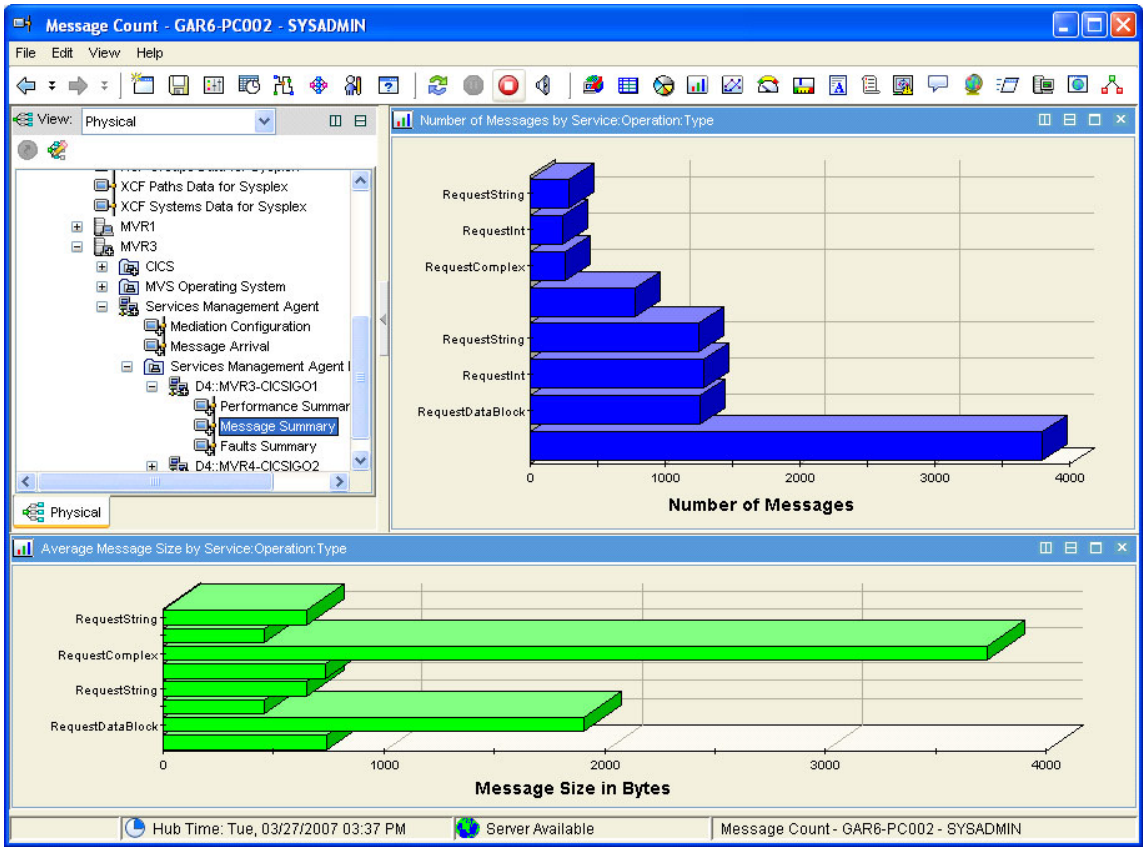


Figure 7-36 ITCAM for SOA CICS Agent - Message Summary

Figure 7-36 on page 225 shows the message count and message size of different operations of a CICS Web service.

Another important factor in the performance of a Web service is the type of data that is transported in the SOAP message. In this test, the response messages of the CICS service provider application contain different data types (strings, integers, complex types, and so on).

While Tivoli OMEGAMON XE for CICS gives a CICS specific view of Web service health, ITCAM for SOA is designed to monitor Web services across the different platforms of an SOA environment. One of ITCAM for SOA's primary advantages is its ability to monitor a variety of different Web services runtime environments, including WebSphere Application Server, CICS, and, most recently, WebSphere DataPower. This helps IT operators to identify the source of problems across the IT infrastructure using a single console.

Figure 7-37 shows a Performance Summary view as reported by the DataPower agent of the ITCAM for SOA.

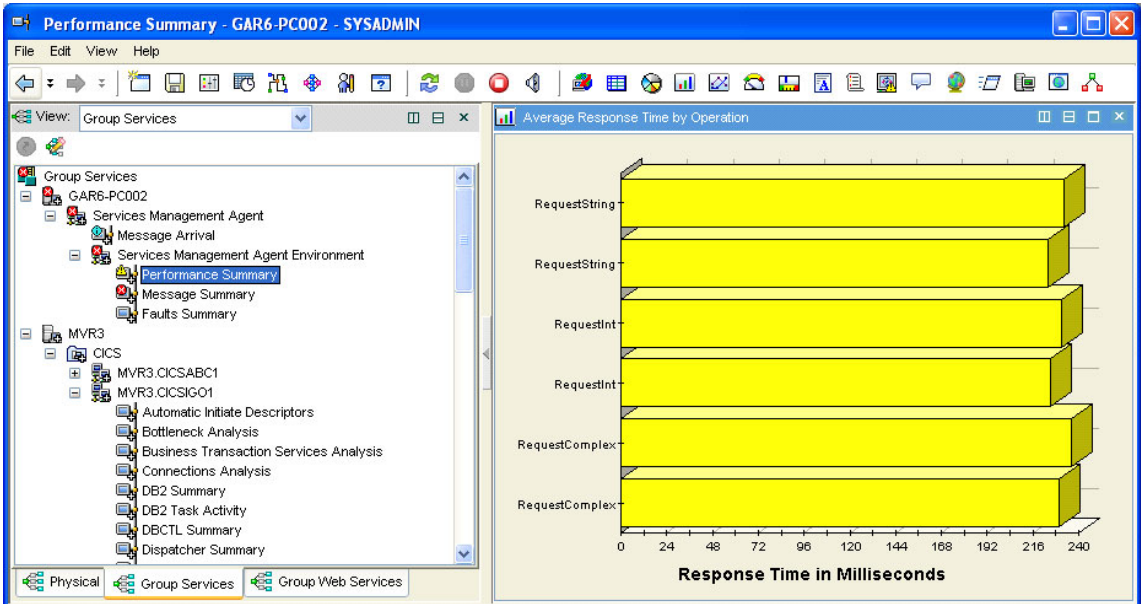


Figure 7-37 ITCAM for SOA DataPower Agent - Performance Summary

Similar information can be retrieved from WebSphere Application Server and CICS, providing the operator with a system wide view of service response times.

For other Tivoli monitoring scenarios and for general information about monitoring an SOA infrastructure, we recommend that you refer to *Best Practices for SOA Management*, REDP-4233.

7.7 Summary

This section describes a CICS Web services configuration based on the non-functional requirements of a particular customer. The example demonstrates that it is possible to embrace important strategic initiatives such as SOA, and at the same time continue to derive value from tried and trusted CICS applications.

A

Sample REXX scripts

In Example A-1 through Example A-5 on page 239, we list the sample REXX coding we used.

Example: A-1 Sample REXX code to copy defintions from one CICSplex to another

```
/* REXX */
/*********************************************************************/
/*                                                                   */
/* Use API to copy a resource definiton from one cicsplex to another. */
/*                                                                   */
/* Restrictions: 1. The CICSPlex must be managed by the same CMAS     */
/*               2. The RESGROUP must already exist                  */
/*               3. The exact same defintion (including version)     */
/*                  is being copied.                                 */
/*                                                                   */
/* Input: RESOURCE, RESGROUP, RESTYPE                                */
/*                                                                   */
/*********************************************************************/
Trace off
/*                                                                 */
parse upper arg W_RESOURCE ',' W_RESGROUP ',' W_RESTYPE .
Parse Value 0 0 0 With W_RESPONSE W_REASON rc .

If W_RASGNDEF ='' | W_RESOURCE ='' then Signal NO_PARM
/* If W_RESTYPE = 'TRANDEF' Then W_INTO_OBJECTLEN = 536 */
If W_RESTYPE = 'TRANDEF' Then W_INTO_OBJECTLEN = 544
Else If W_RESTYPE = 'FILEDEF' Then W_INTO_OBJECTLEN = 424
```

```
Else Signal INV_RESTYPE

W_CONTEXT_FROM = 'SC66PLEX'
W_SCOPE_FROM = 'SC66PLEX'
W_CONTEXT_TO = 'SC65PLEX'
W_SCOPE_TO   = 'SC65PLEX'
W_RESOURCE = left(W_RESOURCE,8)     /* pad the resource to 8 Bytes */

XX = EYUINIT()
If XX <> 0 Then Signal UNEXPECTED

Call CONNECT W_CONTEXT_FROM W_SCOPE_FROM

/*-------------------------------------------------------------------*/
/*    Check the Resource group is valid                              */
/*-------------------------------------------------------------------*/

W_CRITERIA = 'RESGROUP='|| W_RESGROUP || '.'
W_CRITERIALEN = length(W_CRITERIA)

XX = EYUAPI('GET OBJECT(RESGROUP)' ,
                'RESULT(W_RESULT)' ,
                'CRITERIA(W_CRITERIA)' ,
                'LENGTH('W_CRITERIALEN')' ,
                'THREAD(W_THREAD)' ,
                'RESPONSE(W_RESPONSE)' ,
                'REASON(W_REASON)')
If XX <> 0 Then Signal UNEXPECTED
If W_RESPONSE=EYURESP(NODATA) Then Signal NO_RESGROUP
If W_RESPONSE <> EYURESP(OK) Then Signal NO_GET

/*-------------------------------------------------------------------*/
/*    Get the resource defintion being copied                        */
/*-------------------------------------------------------------------*/
W_CRITERIA = 'NAME=' || W_RESOURCE || '.'
W_CRITERIALEN = length(W_CRITERIA)

XX = EYUAPI('GET OBJECT('W_RESTYPE')' ,
                'RESULT(W_RESULT)' ,
                'CRITERIA(W_CRITERIA)' ,
                'LENGTH('W_CRITERIALEN')' ,
                'THREAD(W_THREAD)' ,
                'RESPONSE(W_RESPONSE)' ,
                'REASON(W_REASON)')
If XX <> 0 Then Signal UNEXPECTED
If W_RESPONSE=EYURESP(NODATA) Then Signal NO_RESOURCE
If W_RESPONSE <> EYURESP(OK) Then Signal NO_GET

XX = EYUAPI('FETCH INTO(W_INTO_OBJECT)' ,
```

```
                  'LENGTH(W_INTO_OBJECTLEN)' ,
                  'RESULT(W_RESULT)' ,
                  'THREAD(W_THREAD)' ,
                  'RESPONSE(W_RESPONSE)' ,
                  'REASON(W_REASON)')
If XX <> 0 Then Signal UNEXPECTED
If W_RESPONSE <> EYURESP(OK) Then Signal NO_FETCH

XX = EYUAPI('TPARSE OBJECT('W_RESTYPE')' ,
                   'PREFIX(RES)' ,
                   'STATUS(W_RESPONSE)' ,
                   'VAR(W_INTO_OBJECT.1)' ,
                   'THREAD(W_THREAD)')

If XX <> 0 Then Signal UNEXPECTED
If W_RESPONSE <> 'OK' Then Signal NO_TPARSE

/*-------------------------------------------------------------------*/
/*    Disconect and reconnect to the New CICSPlex                    */
/*-------------------------------------------------------------------*/

Call TERMINATE
Call CONNECT W_CONTEXT_TO W_SCOPE_TO

RES_NAME = 'SOAA    '           /* just for testing */

/*-------------------------------------------------------------------*/
/*    Create the new defintion                                       */
/*-------------------------------------------------------------------*/

XX = EYUAPI('TBUILD OBJECT('W_RESTYPE')' ,
                   'PREFIX(RES)' ,
                   'STATUS(W_RESPONSE)' ,
                   'VAR(W_NEW_OBJECT)' ,
                   'THREAD(W_THREAD)')

If XX <> 0 Then Signal UNEXPECTED
If W_RESPONSE <> 'OK' Then Signal NO_TBUILD

W_PARMS = 'RESGROUP('W_RESGROUP || ').'
W_PARMSLEN = length(W_PARMS)

XX = EYUAPI('CREATE OBJECT('W_RESTYPE')' ,
                'FROM(W_NEW_OBJECT)' ,
                'LENGTH('W_INTO_OBJECTLEN')' ,
                'PARM(W_PARMS) PARMLEN('W_PARMSLEN')' ,
                'THREAD(W_THREAD)' ,
                'RESPONSE(W_RESPONSE)' ,
                'REASON(W_REASON)')
```

```
If XX <> 0 Then Signal UNEXPECTED
If W_RESPONSE <> EYURESP(OK) Then Signal NO_CREATE

Call TERMINATE
Signal ENDIT

CONNECT:
/*-------------------------------------------------------------------*/
/*    Connect to CICSPlex SM                                         */
/*-------------------------------------------------------------------*/

Parse arg W_CONTEXT W_SCOPE

XX = EYUAPI('CONNECT' ,
            'CONTEXT('W_CONTEXT')' ,
            'SCOPE('W_SCOPE')' ,
            'VERSION(0310)' ,
            'THREAD(W_THREAD)' ,
            'RESPONSE(W_RESPONSE)' ,
            'REASON(W_REASON)')
If XX <> 0 Then Signal UNEXPECTED
If W_RESPONSE <> EYURESP(OK) Then Signal NO_CONNECT

Return

TERMINATE:
/*-------------------------------------------------------------------*/
/*    Terminate API Connection                                       */
/*-------------------------------------------------------------------*/

XX = EYUAPI('TERMINATE RESPONSE(W_RESPONSE) REASON(W_RESASON)')

Return

/*-------------------------------------------------------------------*/
/*    PROCESSING FOR API FAILURES.                                   */
/*-------------------------------------------------------------------*/
UNEXPECTED:
      W_MSG_TEXT = 'Unexpected Error.'
      Signal SCRNLOG
NO_PARM:
      W_MSG_TEXT = 'Error - Parameters not passed.'
      Signal SCRNLOG
INV_RESTYPE:
      W_MSG_TEXT = 'Error - Resource Type is invailid.'
      Signal SCRNLOG
NO_CONNECT:
      W_MSG_TEXT = 'Error connecting to API.'
```

```
      Signal SCRNLOG
NO_GET:
      W_MSG_TEXT = 'Error getting RASGNDEF Table.'
      Signal SCRNLOG
NO_CREATE:
      W_MSG_TEXT = 'Error creating the definition.'
      Signal SCRNLOG
NO_TBUILD:
      W_MSG_TEXT = 'Error tbuilding the result set.'
      Signal SCRNLOG
NO_TPARSE:
      W_MSG_TEXT = 'Error tparsing the result set.'
      Signal SCRNLOG
NO_RESOURCE:
      W_MSG_TEXT = 'Resource' W_RESOURCE 'Not Found.'
      Signal SCRNLOG
NO_RESGROUP:
      W_MSG_TEXT = 'Resource group' W_RESGROUP 'Not Found.'
      Signal SCRNLOG
NO_FETCH:
      W_MSG_TEXT = 'Error fetching result set.'
      Signal SCRNLOG
SCRNLOG:
      rc = 8
      Say W_MSG_TEXT
      Say 'RESPONSE='||W_RESPONSE ,
          'REASON='||W_REASON 'RESULT='XX
ENDIT:
XX = EYUTERM()
Exit rc
```

Example: A-2 Sample REXX code to install a resource

```
/* REXX */
/*********************************************************************/
/*                                                                   */
/* Use API to install a resource definition                          */
/*                                                                   */
/* Input: RASGNDEF, RESOURCE                                         */
/*                                                                   */
/*********************************************************************/
Trace off
/*                                                                 */
parse upper arg W_RASGNDEF ',' W_RESOURCE ',' .
If W_RASGNDEF ='' | W_RESOURCE ='' then Signal NO_PARM
/*-------------------------------------------------------------------*/
/*    Get an API connection                                          */
/*-------------------------------------------------------------------*/
Parse Value 0 0 0 With W_RESPONSE W_REASON rc .
```

```
W_CONTEXT = 'SC66PLEX'
W_SCOPE = 'SC66PLEX'
W_RESOURCE = left(W_RESOURCE,8)     /* pad the resource to 8 Bytes */

XX = EYUINIT()
If XX <> 0 Then Signal UNEXPECTED
XX = EYUAPI('CONNECT' ,
            'CONTEXT('W_CONTEXT')' ,
            'SCOPE('W_SCOPE')' ,
            'VERSION(0310)' ,
            'THREAD(W_THREAD)' ,
            'RESPONSE(W_RESPONSE)' ,
            'REASON(W_REASON)')
If XX <> 0 Then Signal UNEXPECTED
If W_RESPONSE <> EYURESP(OK) Then Signal NO_CONNECT
/*-------------------------------------------------------------------*/
/*    Use the RASGNDEF to get the group for the resource          */
/*-------------------------------------------------------------------*/
W_CRITERIA = 'RESASSGN =' || W_RASGNDEF || '.'
W_CRITERIALEN = length(W_CRITERIA)
XX = EYUAPI('GET OBJECT(RASGNDEF)' ,
               'RESULT(W_RESULT)' ,
               'CRITERIA(W_CRITERIA)' ,
               'LENGTH('W_CRITERIALEN')' ,
               'THREAD(W_THREAD)' ,
               'RESPONSE(W_RESPONSE)' ,
               'REASON(W_REASON)')
If XX <> 0 Then Signal UNEXPECTED
If W_RESPONSE=EYURESP(NODATA) Then Signal NO_RASGNDEF
If W_RESPONSE <> EYURESP(OK) Then Signal NO_GET

W_INTO_OBJECTLEN = 1248
XX = EYUAPI('FETCH INTO(W_INTO_OBJECT)' ,
                 'LENGTH(W_INTO_OBJECTLEN)' ,
                 'RESULT(W_RESULT)' ,
                 'THREAD(W_THREAD)' ,
                 'RESPONSE(W_RESPONSE)' ,
                 'REASON(W_REASON)')
If XX <> 0 Then Signal UNEXPECTED
If W_RESPONSE <> EYURESP(OK) Then Signal NO_FETCH
XX = EYUAPI('TPARSE OBJECT(RASGNDEF)' ,
                  'PREFIX(RSGN)' ,
                  'STATUS(W_RESPONSE)' ,
                  'VAR(W_INTO_OBJECT.1)' ,
                  'THREAD(W_THREAD)')

If W_RESPONSE <> 'OK' Then Signal NO_TPARSE
```

```
W_CRITERIA = 'RESGROUP='RSGN_RESGROUP || '.'
W_CRITERIALEN = length(W_CRITERIA)
W_PARMS = 'ASSIGNMENT('W_RASGNDEF'),FORCEINS(YES),' ,
          'CRITERIA(NAME='W_RESOURCE || '.).'
W_PARMSLEN = length(W_PARMS)

XX = EYUAPI('PERFORM OBJECT(RESGROUP)' ,
                'ACTION(INSTALL)' ,
                'RESULT(W_RESULT)' ,
                'CRITERIA(W_CRITERIA)' ,
                'LENGTH('W_CRITERIALEN')' ,
                'PARM(W_PARMS) PARMLEN('W_PARMSLEN')' ,
                'THREAD(W_THREAD)' ,
                'RESPONSE(W_RESPONSE)' ,
                'REASON(W_REASON)')

If W_RESPONSE <> EYURESP(OK) Then Signal NO_INSTALL

Signal ENDIT
/*-------------------------------------------------------------------*/
/*    PROCESSING FOR API FAILURES.                                   */
/*-------------------------------------------------------------------*/
UNEXPECTED:
      W_MSG_TEXT = 'Unexpected Error.'
      Signal SCRNLOG
NO_PARM:
      W_MSG_TEXT = 'Error - Parameters not passed.'
      Signal SCRNLOG
NO_CONNECT:
      W_MSG_TEXT = 'Error connecting to API.'
      Signal SCRNLOG
NO_GET:
      W_MSG_TEXT = 'Error getting RASGNDEF Table.'
      Signal SCRNLOG
NO_INSTALL:
      W_MSG_TEXT = 'Error Installing the definition.'
      Signal SCRNLOG
NO_TPARSE:
      W_MSG_TEXT = 'Error tparsing the result set.'
      Signal SCRNLOG
NO_RASGNDEF:
      W_MSG_TEXT = 'RASGNDEF' W_RASGNDEF 'Not Found.'
      Signal SCRNLOG
NO_FETCH:
      W_MSG_TEXT = 'Error fetching result set.'
      Signal SCRNLOG
SCRNLOG:
      rc = 8
      Say W_MSG_TEXT
```

```
      Say 'RESPONSE='||W_RESPONSE ,
          'REASON='||W_REASON 'RESULT='XX
ENDIT:
/*-------------------------------------------------------------------*/
/*    TERMINATE API CONNECTION.                                      */
/*-------------------------------------------------------------------*/
XX = EYUAPI('TERMINATE RESPONSE(W_RESPONSE) REASON(W_RESASON)')
XX = EYUTERM()
Exit rc
```

Example: A-3 Sample REXX code to phase in a new program

```
/* REXX */
/*********************************************************************/
/*                                                                   */
/* Use API to phasein a program                                      */
/*                                                                   */
/* Input: W_CONTEXT, W_SCOPE and W_PROGRAM                           */
/*                                                                   */
/*********************************************************************/
Trace off

parse upper arg W_CONTEXT ',' W_SCOPE ',' W_PROGRAM .
Parse Value 0 0 0 With W_RESPONSE W_REASON rc .
If W_CONTEXT='' | W_SCOPE='' | W_PROGRAM='' then Signal NO_PARM

/*-------------------------------------------------------------------*/
/*    Connect to CICSPlex SM                                         */
/*-------------------------------------------------------------------*/

XX = EYUINIT()
If XX <> 0 Then Signal UNEXPECTED

XX = EYUAPI('CONNECT' ,
            'CONTEXT('W_CONTEXT')' ,
            'SCOPE('W_SCOPE')' ,
            'VERSION(0310)' ,
            'THREAD(W_THREAD)' ,
            'RESPONSE(W_RESPONSE)' ,
            'REASON(W_REASON)')
If XX <> 0 Then Signal UNEXPECTED
If W_RESPONSE <> EYURESP(OK) Then Signal NO_CONNECT

/*-------------------------------------------------------------------*/
/*    Refresh the program                                            */
/*-------------------------------------------------------------------*/
W_CRITERIA = 'PROGRAM=' || W_PROGRAM || '.'
W_CRITERIALEN = length(W_CRITERIA)
```

```
XX = EYUAPI('PERFORM OBJECT(PROGRAM) ACTION(PHASEIN)',
                       'THREAD(W_THREAD)',
                       'CRITERIA(W_CRITERIA)' ,
                       'LENGTH('W_CRITERIALEN')' ,
                       'RESULT(W_RESTOK)',
                       'RESPONSE(W_RESPONSE) REASON(W_REASON)')

If XX <> 0 Then Signal UNEXPECTED
If W_RESPONSE=EYURESP(NOTAVAILABLE) & W_REASON=EYUREAS(SCOPE) Then
   Signal SCOPE_NOT_AVA
If W_RESPONSE=EYURESP(NODATA) Then Signal NO_DATA
If W_RESPONSE <> EYURESP(OK) Then Signal NO_REFRESH

Say 'Program:' W_PROGRAM 'Has been refreshed'

Signal ENDIT
/*-------------------------------------------------------------------*/
/*    PROCESSING FOR API FAILURES.                                   */
/*-------------------------------------------------------------------*/
UNEXPECTED:
      W_MSG_TEXT = 'Unexpected error.'
      Signal SCRNLOG
NO_PARM:
      W_MSG_TEXT = 'Error - Parameters not passed.'
      Signal SCRNLOG
NO_CONNECT:
      W_MSG_TEXT = 'Error connecting to API.'
      Signal SCRNLOG
NO_DATA:
      W_MSG_TEXT = 'No program available for this scope.'
      Say W_MSG_TEXT
      Signal ENDIT
NO_REFRESH:
      W_MSG_TEXT = 'Error on phasein of program '|| W_PROGRAM
      Signal SCRNLOG
SCOPE_NOT_AVA:
      W_MSG_TEXT = 'CICS region(s) are not available.'
      Say W_MSG_TEXT
      Signal ENDIT
SCRNLOG:
      rc = 8
      Say W_MSG_TEXT
      Say 'RESPONSE='||W_RESPONSE ,
          'REASON='||W_REASON 'RESULT='XX
ENDIT:
/*-------------------------------------------------------------------*/
/*    TERMINATE API CONNECTION.                                      */
/*-------------------------------------------------------------------*/
XX = EYUAPI('TERMINATE RESPONSE(W_RESPONSE) REASON(W_REASON)')
```

```
XX = EYUTERM()
Exit rc
```

Example: A-4 Sample REXX code to phase out a JVM

```
/* REXX */
/*********************************************************************/
/*                                                                   */
/* Use API to phaseout a JVM so that the java is refreshed           */
/*                                                                   */
/* Input: W_CONTEXT, W_SCOPE                                         */
/*                                                                   */
/*********************************************************************/
Trace off

parse upper arg W_CONTEXT ',' W_SCOPE .
Parse Value 0 0 0 With W_RESPONSE W_REASON rc .
If W_CONTEXT='' | W_SCOPE='' then Signal NO_PARM

/*-------------------------------------------------------------------*/
/*    Connect to CICSPlex SM                                         */
/*-------------------------------------------------------------------*/

XX = EYUINIT()
If XX <> 0 Then Signal UNEXPECTED

XX = EYUAPI('CONNECT' ,
            'CONTEXT('W_CONTEXT')' ,
            'SCOPE('W_SCOPE')' ,
            'VERSION(0310)' ,
            'THREAD(W_THREAD)' ,
            'RESPONSE(W_RESPONSE)' ,
            'REASON(W_REASON)')
If XX <> 0 Then Signal UNEXPECTED
If W_RESPONSE <> EYURESP(OK) Then Signal NO_CONNECT

/*-------------------------------------------------------------------*/
/*    Refresh the JVM's                                              */
/*-------------------------------------------------------------------*/

XX = EYUAPI('PERFORM OBJECT(JVMPOOL) ACTION(PHASEOUT)',
                      'THREAD(W_THREAD)',
                      'RESULT(W_RESTOK)',
                      'RESPONSE(W_RESPONSE) REASON(W_REASON)')

If XX <> 0 Then Signal UNEXPECTED
If W_RESPONSE=EYURESP(NOTAVAILABLE) & W_REASON=EYUREAS(SCOPE) Then
   Signal SCOPE_NOT_AVA
If W_RESPONSE=EYURESP(NODATA) Then Signal NO_DATA
```

```
If W_RESPONSE <> EYURESP(OK) Then Signal NO_REFRESH

Say 'JVMs have been refreshed (phased out)'

Signal ENDIT
/*-------------------------------------------------------------------*/
/*    PROCESSING FOR API FAILURES.                                   */
/*-------------------------------------------------------------------*/
UNEXPECTED:
      W_MSG_TEXT = 'Unexpected error.'
      Signal SCRNLOG
NO_PARM:
      W_MSG_TEXT = 'Error - Parameters not passed.'
      Signal SCRNLOG
NO_CONNECT:
      W_MSG_TEXT = 'Error connecting to API.'
      Signal SCRNLOG
NO_DATA:
      W_MSG_TEXT = 'No JVMs available for this scope.'
      Say W_MSG_TEXT
      Signal ENDIT
NO_REFRESH:
      W_MSG_TEXT = 'Error on phaseout of the JVMs.'
      Signal SCRNLOG
SCOPE_NOT_AVA:
      W_MSG_TEXT = 'CICS region(s) are not available.'
      Say W_MSG_TEXT
      Signal ENDIT
SCRNLOG:
      rc = 8
      Say W_MSG_TEXT
      Say 'RESPONSE='||W_RESPONSE ,
          'REASON='||W_REASON 'RESULT='XX
ENDIT:
/*-------------------------------------------------------------------*/
/*    TERMINATE API CONNECTION.                                      */
/*-------------------------------------------------------------------*/
XX = EYUAPI('TERMINATE RESPONSE(W_RESPONSE) REASON(W_REASON)')
XX = EYUTERM()
Exit rc
```

Example: A-5 Sample REXX code to perfrom a pipe line scan

```
/*********************************************************************/
/*                                                                   */
/* Use API to perform a pipe line scan to refresh the web services   */
/* from the HFS                                                      */
/*                                                                   */
/* Input: W_CONTEXT, W_SCOPE, W_PIPE                                 */
```

```
/*                                                                   */
/*********************************************************************/
Trace off

parse upper arg W_CONTEXT ',' W_SCOPE ',' W_PIPE .
Parse Value 0 0 0 With W_RESPONSE W_REASON rc .
If W_CONTEXT='' | W_SCOPE='' | W_PIPE='' then Signal NO_PARM

/*-------------------------------------------------------------------*/
/*    Connect to CICSPlex SM                                         */
/*-------------------------------------------------------------------*/

XX = EYUINIT()
If XX <> 0 Then Signal UNEXPECTED

XX = EYUAPI('CONNECT' ,
            'CONTEXT('W_CONTEXT')' ,
            'SCOPE('W_SCOPE')' ,
            'VERSION(0310)' ,
            'THREAD(W_THREAD)' ,
            'RESPONSE(W_RESPONSE)' ,
            'REASON(W_REASON)')
If XX <> 0 Then Signal UNEXPECTED
If W_RESPONSE <> EYURESP(OK) Then Signal NO_CONNECT

/*-------------------------------------------------------------------*/
/*    Refresh the Pipeline                                           */
/*-------------------------------------------------------------------*/
W_CRITERIA = 'NAME=' || W_PIPE || '.'
W_CRITERIALEN = length(W_CRITERIA)

XX = EYUAPI('PERFORM OBJECT(PIPELINE) ACTION(SCAN)',
                       'THREAD(W_THREAD)',
                       'CRITERIA(W_CRITERIA)' ,
                       'LENGTH('W_CRITERIALEN')' ,
                       'RESULT(W_RESTOK)',
                       'RESPONSE(W_RESPONSE) REASON(W_REASON)')

If XX <> 0 Then Signal UNEXPECTED
If W_RESPONSE=EYURESP(NOTAVAILABLE) & W_REASON=EYUREAS(SCOPE) Then
   Signal SCOPE_NOT_AVA
If W_RESPONSE=EYURESP(NODATA) Then Signal NO_DATA
If W_RESPONSE <> EYURESP(OK) Then Signal NO_REFRESH

Say 'Pipe line:' W_PIPE 'Has been refreshed'

Signal ENDIT
/*-------------------------------------------------------------------*/
/*    PROCESSING FOR API FAILURES.                                   */
```

```
/*-------------------------------------------------------------------*/
UNEXPECTED:
      W_MSG_TEXT = 'Unexpected error.'
      Signal SCRNLOG
NO_PARM:
      W_MSG_TEXT = 'Error - Parameters not passed.'
      Signal SCRNLOG
NO_CONNECT:
      W_MSG_TEXT = 'Error connecting to API.'
      Signal SCRNLOG
NO_DATA:
      W_MSG_TEXT = 'No pipeline available for this scope.'
      Say W_MSG_TEXT
      Signal ENDIT
NO_REFRESH:
      W_MSG_TEXT = 'Error on scan of pipe line '|| W_PIPE
      Signal SCRNLOG
SCOPE_NOT_AVA:
      W_MSG_TEXT = 'CICS region(s) are not available.'
      Say W_MSG_TEXT
      Signal ENDIT
SCRNLOG:
      rc = 8
      Say W_MSG_TEXT
      Say 'RESPONSE='||W_RESPONSE ,
          'REASON='||W_REASON 'RESULT='XX
ENDIT:
/*-------------------------------------------------------------------*/
/*    TERMINATE API CONNECTION.                                      */
/*-------------------------------------------------------------------*/
XX = EYUAPI('TERMINATE RESPONSE(W_RESPONSE) REASON(W_REASON)')
XX = EYUTERM()
Exit rc
```

B

A useful header processing program

In this section, we provide an example of a header processing program that we use to monitor the contents of containers at the end of the pipeline. We found this tool very useful for debugging.

The default CICS-provided pipeline configuration file basicsoap11provider.xml was copied from the installation HFS directory to a private directory and modified, as shown in Example B-1.

Example: B-1 Modified basicsoap11provider.xml file

```
<?xml version="1.0" encoding="UTF-8"?>
<provider_pipeline xmlns="http://www.ibm.com/software/htp/cics/pipeline">
  <service>
    <terminal_handler>
      <cics_soap_1.2_handler>
       <headerprogram>
         <program_name>SHOWINFO</program_name>
         <namespace>http://bogus/name/space</namespace>
         <localname>somename</localname>
         <mandatory>true</mandatory>
       </headerprogram>
      </cics_soap_1.2_handler>
    </terminal_handler>
  </service>
```

```
  <apphandler>DFHPITP</apphandler>
</provider_pipeline>
```

We first changed the value of the encoding parameter on the first line of the file from the CICS-provided value of “EBCDIC-CP-US”? to the value “UTF-8”?. The reason for that was that we were debugging communication between a SOAP client program running in WebSphere Application Server. The CICS-provided example assumes that both the client and the server parts of the example catalog management application execute under CICS, and that the Web service is invoked by using transaction ECLI.

The <service> element identifies the handlers (one in this case) to be invoked. The <terminal handler> element identifies the message handler at the end of the pipeline, which is the CICS default SOAP handler for SOAP 1.2. This handler also works with SOAP 1.1. The default CICS SOAP handler will not find any headers with the specified namespace (http://bogus/name/space) and localname (somename), but because mandatory is set to true, the SHOWINFO handler will be invoked. The program browses all the containers on the channel that is passed to it and writes out their names and contents to CSMT transient data destination. The default CICS SOAP handler then invokes the program specified in the <apphandler> element, which happens to be the CICS-provided DFHPITP. This program prepares, based on rules in the WSBIND file, the data for the user program that expects to get it either in a COMMAREA or from a CHANNEL. DFHPITP then invokes the user program, the name of which is specified in the WSBIND file.

Example B-2 shows the source of the SHOWINFO program. Make sure to link-edit this program with AMODE(31) because the CPIH transaction under which the program runs is defined with the attribute TASKDATAALLOC(ANY).

Example: B-2 SHOWINFO message handler program

```
CBL CICS('COBOL3') APOST TRUNC(OPT) NOADATA
       IDENTIFICATION DIVISION.
       PROGRAM-ID. SHOWINFO.
      *
      * This is a sample CICS TS V3.2 message handler
      *  This program prints the name of the channel and
      *  the names of the containers in the channel.
      *
       DATA DIVISION.
       WORKING-STORAGE SECTION.
       01  WS-VARIABLES.
           05  Pgm-Name               PIC X(9) Value 'SHOWINFO-'.
           05  STRT-MSG               PIC X(30)
                  VALUE 'Start of pgm --------'.
           05  END-MSG                PIC X(30)
```

```
             VALUE 'End of pgm ----------'.
       05  Channel-Msg.
           10  FILLER              PIC X(14)
                 VALUE 'Channel name: '.
           10  Channel-Name        PIC X(16).
       05  Container-Msg.
           10  FILLER              PIC X(16)
                 VALUE 'Container name: '.
           10  Container-Name      PIC X(16).
           10  FILLER              PIC X(9)
                 VALUE ', Length='.
           10  Container-Len-Msg   PIC zzzzzzzz9.
       05  Container-Length        PIC S9(8) Comp.
       05  start-Index             PIC S9(8) Comp.
       05  size-Index               PIC S9(8) Comp.
       05  Resp-Fld                PIC S9(8) Comp.
       05  Resp2-Fld               PIC S9(8) Comp.
       05  TOKEN                   PIC S9(8) COMP-5 SYNC.
       05  Abs-Time                PIC S9(15) COMP-3.
       05  The-Date                PIC X(8).
       05  The-Time                PIC X(8).
       05  CSMT-Output-Buffer      PIC X(121).
       05  Message-to-Write        PIC X(90).
       05  More-Containers         PIC X(3) Value 'YES'.
           88  No-More-Containers           Value 'NO'.
       05  Data-End                PIC X(3) Value 'No '.
           88  End-Of-Data                  Value 'Yes'.

      *
       LINKAGE SECTION.
       01  The-Container-Contents      PIC X.
      *
       PROCEDURE DIVISION.

           Perform Write-Start-Msg.

           EXEC CICS ASSIGN CHANNEL(Channel-Name) END-EXEC.

           Move Channel-Msg to Message-To-Write.
           Perform Write-to-CSMT-Queue.

           EXEC CICS STARTBROWSE CONTAINER
                     CHANNEL(Channel-Name)
                     BROWSETOKEN(TOKEN)
                     RESP(Resp-Fld)
                     RESP2(Resp2-Fld)
                     END-EXEC.

           If Resp-Fld NOT = DFHRESP(NORMAL)
```

```
            Move 'Error requesting start browse on channel'
                to Message-to-Write
            Perform Write-to-CSMT-Queue
            Perform Write-End-Msg-and-Return
        End-If.

        Perform Write-Container-Name
            Until No-More-Containers.

        Perform Write-End-Msg-and-Return.


      * ******************************************************
      *  Get Next Container name and write it
      * ******************************************************
       Write-Container-Name.
        EXEC CICS GETNEXT CONTAINER(Container-Name)
                  BROWSETOKEN(TOKEN)
                  RESP(Resp-Fld)
                  RESP2(Resp2-Fld)
                  END-EXEC.
        Evaluate Resp-Fld
            WHEN DFHRESP(NORMAL)
                Move 0 to Container-Length
                EXEC CICS GET CONTAINER(Container-Name)
                    NODATA FLENGTH(Container-Length)
                    RESP(Resp-Fld)
                    RESP2(Resp2-FLD)
                    END-EXEC
                Move Container-Length to Container-Len-Msg
                Move Container-Msg to Message-to-Write
                Perform Write-to-CSMT-Queue
                If Container-Length NOT = 0
                    Perform Write-Container-Contents
                End-If
            WHEN DFHRESP(END)
                Move 'NO' to More-Containers
            WHEN OTHER
                Move 'Error requesting next container name'
                    to Message-to-Write
                Perform Write-to-CSMT-Queue
                Move 'NO' to More-Containers
        End-Evaluate.

       Write-Container-Contents.
        EXEC CICS GET CONTAINER(Container-Name)
            SET(ADDRESS OF The-Container-Contents)
            FLENGTH(Container-Length)
            RESP(Resp-Fld)
```

```
            RESP2(Resp2-FLD)
            END-EXEC.
        If Resp-Fld = DFHRESP(NORMAL)
            Move 1 to start-Index
            Move 'No' to Data-End
            Perform Write-Line  Until End-Of-Data
        End-If.

 Write-Line.
        Move 110 to size-Index.
        If ((start-Index + size-Index) > Container-Length)
            Compute size-Index = Container-Length - start-Index + 1
            Move 'Yes' to Data-End
        End-If.

        Move SPACE to CSMT-Output-Buffer.
        String ' ---> '
            The-Container-Contents(start-Index:size-Index)
           Delimited by Size
        Into CSMT-Output-Buffer.
        EXEC CICS WRITEQ TD QUEUE('CSMT')
            FROM (CSMT-Output-Buffer)
            LENGTH(LENGTH OF CSMT-Output-Buffer)
            RESP(Resp-Fld)
            END-EXEC.
        Compute start-Index = start-Index + 110.


* ******************************************************
*  Program Start and End Messages
* ******************************************************
 Write-Start-Msg.
        Move Strt-Msg to Message-To-Write.
        Perform Write-to-CSMT-Queue.

 Write-End-Msg-and-Return.
        Move End-Msg to Message-To-Write.
        Perform Write-to-CSMT-Queue.
        EXEC CICS RETURN END-EXEC.

* ******************************************************
*  The WRITE-TO-CSMT-QUEUE  gets the current date and time
*  along with the value placed in Message-To-Write and
*  writes that to the CSMT queue.  If anything goes
*  wrong during the TD queue write, it will be ignored
* ******************************************************
 Write-to-CSMT-Queue.
        Move SPACE to CSMT-Output-Buffer.
        PERFORM Get-and-Format-Current-Time.
```

```
            String Pgm-Name
                The-Date '-' The-Time '-'
                Message-To-Write Delimited by Size
             Into CSMT-Output-Buffer.
            EXEC CICS WRITEQ TD QUEUE('CSMT')
                FROM (CSMT-Output-Buffer)
                LENGTH(LENGTH OF CSMT-Output-Buffer)
                RESP(Resp-Fld)
                END-EXEC.

        Get-and-Format-Current-Time.
            EXEC CICS ASKTIME ABSTIME(Abs-Time) END-EXEC.
            EXEC CICS FORMATTIME ABSTIME(Abs-Time)
                         DATE(The-Date) DATESEP('/')
                         TIME(The-Time) TIMESEP(':')
                         END-EXEC.
```

C

CICS program as an HTTPS client

CICS V3 provides the capability for an application program to be an HTTP or an HTTPS client. You have to define a URL for the HTTP server that you want to access using a URIMAP resource definition. Alternatively, you can use the EXEC CICS PARSE URL command to build an outbound HTTP request that would be used on a subsequent EXEC CICS OPEN command. You can then use a sequence of EXEC CICS WEB SEND and WEB RECEIVE commands to send and receive requests from HTTP sites. You can use a WEB CONVERSE command instead of WEB SEND and WEB RECEIVE commands. You terminate your session with the Web server by using an EXEC CICS WEB CLOSE command.

In this section, we document the sequence of actions that have to be done in order to enable an application program running in a CICS region to access a secure site using the HTTPS protocol. This sequence was tested with Internet Explorer V6 and V7 and z/OS V1.7.

1. Get a client certificate from the HTTPS site that you plan to access. If the site is accessible with a browser, do the following:
   a. Access the site with Internet Explorer and, when prompted whether you want to accept the certificate, click the **Install** button.
   b. On the Internet Explorer window, select **Tools** → **Internet Options**.

c. Click the **Content** tab.

d. Click the **Certificates** button.

e. Locate the certificate that you want to use and click the **Export** button.

f. Click **Next**.

g. On the next window, click **Base-64 encoded x.509(.CER)** and specify a location for the exported certificate.

h. Click **Next**, **Next**, and **Finish**.

2. Upload the exported certificate as a text file. It will be converted from ASCII to EBCIDIC.

3. Import the certificate into RACF or your security manager product. For RACF, you can use ISPF options or RACDCERT commands. From ISPF:

   a. Enter RACF.

   b. Choose option 7, Digital Certificates and Key Rings.

   c. Choose option 4, Add, Alter, Delete, or List certificates or check whether a digital certificate has been added to the RACF database and associated with a user ID.

   d. Choose option 1 to add the certificate to the RACF database.

   e. Use option 6 to add the new certificate to the key ring that is defined as a SIT option for the CICS region or define a new key ring for CICS.

4. In your application, specify the name of the certificate on the EXEC CICS WEB OPEN command using the CERTIFICATE parameter.

D

Sample XPCREQ global user exit

In order to be able to run the business logic program under the same transaction ID in the back end as in the front end, we had to write a Program Control global user exit to run off XPCREQ.

The reason for this is that the sample application uses program names starting with DFH, and CICS does not allow you to change mirror transaction name for programs starting with DFH.

The user exit might also be used as a model if you have different transaction invoking programs using DPL, and you want these programs to run under the same transaction ID as the caller. This could be for accounting, monitoring, or security reasons.

XPCREQ global user exit

The exit shown in Example 7-5 checks if the program linked to is DFH0XCMN. If so, the transaction ID of the mirror task is set to the transaction ID of the caller.

Example 7-5 Sample XPCREQ global user exit

```
*ASM XOPTS(NOPROLOG,NOEPILOG)
         PRINT    GEN
         DFHUEXIT TYPE=EP,ID=(XPCREQ)
         DFHUEXIT TYPE=XPIENV
         COPY     DFHSAIQY
         COPY     DFHPGISY
         COPY     DFHXMIQY
         COPY     DFHSMMCY
*
         COPY  DFHPCEDS
*
WORK     DSECT
WORKTRAN DS    CL4
*
XPCREQ1  CSECT
XPCREQ1  AMODE 31
XPCREQ1  RMODE ANY
         SAVE  (14,12)                 SAVE REGS
         LR    R11,R15                 SET-UP BASE REGISTER
         USING XPCREQ1,R11
*
         LR    R2,R1                   SET UP ADDRESSABILITY TO
         USING DFHUEPAR,R2             USER EXIT PARM LIST
*
         L     R6,UEPCLPS              SET UP ADDRESSABILTY TO COMMAND
         USING PC_ADDR_LIST,R6         LEVEL PLIST
*
         L     R7,PC_ADDR1             IS PROGRAM NAME = DFHOXCMN ?
         CLC   =C'DFHOXCMN',0(R7)
         BNE   PCEXIT                  NO > EXIT
*
         L     R7,=F'100'              GETMAIN STORAGE
         L     R4,UEPXSTOR
         USING DFHSMMC_ARG,R4
         L     R13,UEPSTACK
*
         DFHSMMCX CALL,                                              +
               CLEAR,                                                +
               IN,                                                   +
FUNCTION(GETMAIN),
               GET_LENGTH((R7)),                                     +
```

```
              SUSPEND(NO),                                          +
              STORAGE_CLASS(USER),                                  +
              OUT,                                                  +
              ADDRESS((R8)),                                        +
              RESPONSE(*),                                          +
              REASON(*)
*
         USING WORK,R8
*
         L     R4,UEPXSTOR             INQUIRE TRANSACTION TO TO
         USING DFHXMIQ_ARG,R4          CURRENT TRANSACTIONS ID
         L     R13,UEPSTACK
*
         DFHXMIQX CALL,                                             +
               CLEAR,                                               +
               IN,                                                  +
               FUNCTION(INQUIRE_TRANSACTION),                       +
               TRANSACTION_TOKEN(*),                                +
               OUT,                                                 +
               TRANSACTION_ID(WORKTRAN),                            +
               RESPONSE(*),                                         +
               REASON(*)
*
         LA    R7,WORKTRAN             SET MIRROR TRANSID = CURRENT
         ST    R7,PC_ADDR8             TRANSID
*
         L     R7,PC_ADDRO             MARK REMOTE TRANSID EXISTS
         USING PC_EID,R7
         OI    PC_BITS1,PC_EXIST8
         DROP  R7
*
         NI    PC_ADDR0,B'01111111'    REMOVE -END-OF-PARMLIST
         NI    PC_ADDR1,B'01111111'
         NI    PC_ADDR2,B'01111111'
         NI    PC_ADDR3,B'01111111'
         NI    PC_ADDR4,B'01111111'
         NI    PC_ADDR5,B'01111111'
NI    PC_ADDR6,B'01111111'
         NI    PC_ADDR7,B'01111111'
*
         TM    PC_ADDR9,B'10000000'
         BO    PCEXIT
*
         TM    PC_ADDRA,B'10000000'
         BO    PCEXIT
*
         OI    PC_ADDR8,B'10000000'    SET END-OF-PARMLIST
*
PCEXIT DS      0H
```

```
         L     R13,UEPEPSA             LOAD ADDRESS OF THE REG SAVE AREA
         RETURN (14,12),RC=UERCNORM    RESTORE REGS AND RETURN NORMAL
*
         END XPCREQ1
/*
```

XPCREQ1 program for enabling global user exit

The program shown in Example 7-6 is used to enable the global user exit for XPCREQ. Typically, you would run this in the PLT stage of CICS startup.

Example 7-6 Program for enabling program XPOCREQ

```
*ASM XOPTS(SP)
*
         DFHREGS
*
DFHEISTG DSECT
*
DFH$PCPI AMODE 31
DFH$PCPI RMODE ANY
DFH$PCPI DFHEIENT
         SPACE
         EXEC CICS ENABLE PROGRAM('XPCREQ') EXIT('XPCREQ') START       +
                         NOHANDLE
*
         END
```

Related publications

The publications listed in this section are considered particularly suitable for a more detailed discussion of the topics covered in this book.

IBM Redbooks

For information about ordering these publications, see “How to get Redbooks” on page 256. Note that some of the documents referenced here may be available in softcopy only.

- *Best Practices for SOA Management*, REDP-4233
- *Implementing CICS Web Services*, SG24-7206
- *Patterns: Service-Oriented Architecture and Web Services*, SG24-6303
- *SOAP Message Size Performance Considerations*, REDP-4344

Other publications

These publications are also relevant as further information sources:

- *CICS Web Services Guide*, SC34-6838
- *CICSPlex SM for z/OS Concepts and Planning,* SC34-6015
- *WebSphere MQ for z/OS Concepts and Planning Guide V5.3.1,* GC34-6051
- *z/OS Communications Server: IP Configuration Reference*, SC31-8776

Online resources

These Web sites are also relevant as further information sources:

- WSDL 2.0 specification

  `http://www.w3.org/TR/wsdl20`
- WSDL specification

  `http://www.w3.org/TR/wsdl`

- XML Description1

  http://www.w3.org/XML/

How to get Redbooks

You can search for, view, or download Redbooks, Redpapers, Technotes, draft publications and Additional materials, as well as order hardcopy Redbooks, at this Web site:

ibm.com/redbooks

Help from IBM

IBM Support and downloads

ibm.com/support

IBM Global Services

ibm.com/services

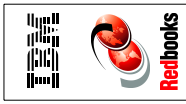
Index

D

E

X

Z



CICS Web Services Workload Management and Availability

(0.5” spine)
0.475”<->0.875”
250 <-> 459 pages

CICS Web Services Workload Management and Availability

Deploying Web services in a CICSplex

Based on CICS TS V3.2 CICSPlex SM

Contains a customer project example

A key feature of CICS TS V3 is its ability to use CICS to provide an execution environment for Web services applications. This IBM Redbooks publication contains the specific management requirements of such applications and shows how CICSPlex SM can be used to manage these applications in an easy to use fashion.

We present an overview of Web services, discuss how CICS implements Web services, discuss the Web services support of CICS TS V3.1 and the enhancements provided with CICS TS V3.2, and give a small overview of CICSPlex SM and how the CICSPlex SM WUI can assist with both CICS Web Services installation and management. We also discuss the different techniques that can be used to provide high system availability and workload management for CICS Web service applications and how high availability is provided across a Parallel Sysplex.

We then describe how CICSPlex SM is used to define and manage the required CICS definitions to run the Web services application and configure CICS as a service provider, show how to set up a CICS Web services requester application for high availability, and describe a practical example of a CICS Web services implementation based on a proof of concept for a large financial group.

INTERNATIONAL TECHNICAL SUPPORT ORGANIZATION

BUILDING TECHNICAL INFORMATION BASED ON PRACTICAL EXPERIENCE

IBM Redbooks are developed by the IBM International Technical Support Organization. Experts from IBM, Customers and Partners from around the world create timely technical information based on realistic scenarios. Specific recommendations are provided to help you implement IT solutions more effectively in your environment.

For more information:
ibm.com/redbooks

SG24-7144-01 ISBN 073848511X